А.В.Матросов

MAPLE 6. РЕШЕНИЕ ЗАДАЧ ВЫСШЕЙ МАТЕМАТИКИ И МЕХАНИКИ

Книга посвящена системе аналитических вычислений Maple 6: представлены все основные понятия языка Maple и наиболее часто используемые функции и объекты; подробно рассмотрены вопросы графического отображения получаемых с помощью Maple решений; дано введение в программирование на языке Maple 6 (ОПП и вызов внешних модулей, написанных на языках высокого уровня); показано использование функций Maple на рабочих листах Excel для проведения аналитических преобразований и вычислений. Особое внимание уделено применению Maple для решения разнообразных задач из курса высшей математики технических университетов, а также задач сопротивления материалов, теории упругости и классической механики. Приводятся полные тексты программ Maple для решения подобных задач.

Содержание

Введение 8
Пример 1: Неопределенный интеграл 9
Пример 2: График функции и поверхности 12
Новое в Maple 6 14
Структура книги 17
Используемые обозначения 18
ЧАСТЬ I. ОСНОВЫ MAPLE **19**
Глава 1. Графический интерфейс пользователя **21**

- 1.1. Общий вид 21
- 1.2. Рабочие листы 24
  - 1.2.1. Область ввода 25
  - 1.2.2. Область вывода 29
  - 1.2.3. Вывод графики 30
- 1.3. Палитры, электронные таблицы, контекстные меню 31
  - 1.3.1. Работа с палитрами 31
  - 1.3.2. Электронная таблица 35
  - 1.3.3. Обмен данными и контекстные меню 41
- 1.4. Работа с меню 45
  - 1.4.1. Структура меню 46
  - 1.4.2. Стандартное меню рабочего листа 47
    - 1.4.2.1. Команды меню File 48
    - 1.4.2.2. Команды меню Edit 50
    - 1.4.2.3. Команды меню View 51
    - 1.4.2.4. Меню Insert, Format, Options и Window 53
- 1.5. Документирование рабочих листов 58
  - 1.5.1. Структурирование документа 58
  - 1.5.2. Работа с несколькими рабочими листами 62
- 1.6. Справочная система 64
  - 1.6.1. Организация справки 65

- 1.6.2. Вызов справки с помощью меню Help 66
- 1.6.3- Вызов справки из рабочего листа 68

Глава 2. Основные объекты и команды 71

- 2.1. Объекты, переменные и выражения 72
  - 2.1.1. Числа 72
  - 2.1.2. Константы 78
  - 2.1.3. Строки 79
  - 2.1.4. Переменные, неизвестные и выражения 81
- 2.2. Команды преобразования выражений 86
  - 2.2.1. Упрощение выражения: simplify() 87
  - 2.2.2. Раскрытие скобок в выражении: expand() 91
  - 2.2.3. Разложение полинома на множители: factor() 92
  - 2.2.4. Сокращение алгебраической дроби: normal() 94
  - 2.2.5. Приведение нескольких членов выражения к одному: combine() 95
  - 2.2.6. Приведение подобных членов: collect() 97
  - 2.2.7. Рационализация дробей: rationalize() 99
  - 2.2.8. Ограничения на неизвестные: assume() 100
- 2.3. Структура выражений и их вычисление 105
  - 2.3.1.Основные сложные типы данных 106
    - 2.3.1.1. Последовательность выражений 106
    - 2.3.1.2. Списки и множества 108
    - 2.3.1.3. Массивы и таблицы 110
  - 2.3.2. Структура выражений и работа с ней 112
    - 2.3.2.1. Структурная обработка списков, множеств и полиномов 113
    - 2.3.2.2. Внутренняя структура выражений 118
    - 2.3.2.3. Подстановка и преобразование типов 124
  - 2.3.3. Вычисление выражений 128
    - 2.3.3.1. Уровни вычислений 128
- 2.4. Решение уравнений, неравенств и их систем 134
  - 2.4.1. Команда solve() 135
  - 2.4.2. Команда fsolve() 141
  - 2.4.3. Другие команды решения уравнений 143
  - 2.4.4. Решение неравенств 144
- 2.5. Дифференцирование и интегрирование 145
- 2.6. Решение обыкновенных дифференциальных уравнений. 149

Глава 3. Пакеты **155**

- 3.1. Организация Maple 155
- 3.2. Линейная алгебра 160
  - 3.2.1. Пакет linalg 161
  - 3.2.2. Пакет LinearAlgebra 168
    - 3.2.2.1. Основные типы данных 169
    - 3.2.2.2. Элементарные операции с матрицами и 174

векторами
- 3.2.2.3. Решение систем линейных уравнений 180
- 3.2.2.4. Вычисления с использованием программ пакета NAG 184
- 3.3. Обыкновенные дифференциальные уравнения 187
- 3.4. Уравнения в частных производных 196
  - 3.4.1. Универсальная команда pdsolve() 197
  - 3.4.2. Команды пакета PDEtools 201
- 3.5. Другие пакеты 207
  - 3.5.1. Пакет student 207
  - 3.5.2. Связь с Matlab 209
  - 3.5.3. Пакет линейной оптимизации simplex 209
  - 3.5.4. Пакет статистики stats 211

Глава 4. Графика 213

- 4.1. Команды двумерной графики 214
  - 4.1.1. Команда plot() 214
  - 4.1.2. Меню для работы с двумерной графикой 222
  - 4.1.3. Двумерные команды пакета plots 224
  - 4.1.4. Двумерные графические структуры Maple 233
  - 4.1.5. Несколько советов 242
- 4.2. Пространственная графика 245
  - 4.2.1. Команда plot3d() 245
  - 4.2.2. Меню для работы с трехмерной графикой 252
  - 4.2.3. Трехмерные команды пакета plots 255
  - 4.2.4. Трехмерные графические структуры Maple 262
- 4.3. Анимация 266
  - 4.3.1. Двумерная анимация 268
  - 4.3.2. Трехмерная анимация 270

Глава 5. Основы программирования в Maple 273

- 5.1. Язык Maple 273
  - 5.1.1. Основные элементы 274
  - 5.1.2. Выражения и типы 281
  - 5.1.3. Операторы 290
- 5.2. Процедуры 297
  - 5.2.1. Определение процедуры 299
  - 5.2.2. Передача параметров 301
  - 5.2.3. Локальные и глобальные переменные 304
  - 5.2.4. Опции и строка описания 308
  - 5.2.5. Возвращаемые значения 313
  - 5.2.6. Объект процедура 321
- 5.3. Работа с файлами 323
- 5.4. Новые возможности Maple 6 330
  - 5.4.1. Модули 331
  - 5.4.2. Вызов внешних процедур 339

Глава 6. Maple в Excel 345
- 6.1. Установка и получение справки 346
- 6.2. Функции Maple на рабочем листе Excel 348
- 6.3. Настройка параметров Maple в Excel 357
- 6.4. Программирование функций Maple в VBA 361

ЧАСТЬ II. МАТЕМАТИКА 367
Глава 7. Аналитическая геометрия и линейная алгебра 369
- 7.1. Аналитическая геометрия на плоскости 369
- 7.2. Аналитическая геометрия в пространстве 379
- 7.3. Линейная алгебра 385

Глава 8. Дифференцирование функций 397
- 8.1. Пределы 397
- 8.2. Производная и ее использование для исследования функций 403

Глава 9. Интегрирование функций 413
- 9.1. Неопределенный интеграл 413
- 9.2. Приложения определенного интеграла 421

Глава 10. Ряды и дифференциальные уравнения 437
- 10.1. Дифференциальные уравнения с разрывными правыми частями 437
- 10.2. Функциональные ряды 442
- 10.3. Приближенное решение дифференциальных уравнений 449
- 10.4. Ряды Фурье 455

Глава 11. Численно-аналитические методы 459
- 11.1. Исследование метода Ньютона 459
- 11.2. Интерполирование функций полиномами 465
- 11.3 Краевые задачи для обыкновенных дифференциальных уравнений 472

ЧАСТЬ III. МЕХАНИКА 479
Глава 12. Задачи теоретической механики 481
Глава 13. Метод начальных параметров в расчете балок 495
Глава 14. Задачи теории упругости 511
Список литературы 522
Предметный указатель 523

Предметный указатель

А

Анимация:
- animate() 268
- animate3d() 270
- display() 269, 271
- формат GIF 269, 271

В

Ввод/вывод:
- readdata() 325
- writedata() 324
- буферизованный и небуферизованный 327
- дескриптор файла 327
- закрытие файла, fclose(), close() 328
- открытие файла, fopen() 327
- открытие файла, open() 328
- текущая позиция файла, filepos() 328
- удаление файла, fremove() 329
- файл, режим доступа 327

файл, текстовый и двоичный 327
Выделение решений, assign() 140
Вызов внешних процедур 339
define_external() 340
дескриптор данных 340
Вызов команд 86
Выражение 82
Вычисление в точке:
eval() 132
evalf() 133
evalhf() 133
subs() 132
Вычисление выражений, value() 86
Вычисление имен 128
assigned() 131
eval() 128
evaln() 130
полное 128
уровень вычисления 128
Г
Графические структуры:
AMBIENLIGHT() 262
AXESLABELS() 235
AXESSTILE() 236
AXESTICKS() 235
COLOR() 236, 263
CURVES() 235
FONT() 236
GRID() 262
GRIDSTYLE() 263
LIGHT() 263
LIGHTMODEL() 263
LINESTYLE() 236
MESH() 262
POINTS() 235
POLYGONS() 235
SCALING() 236
STYLE() 263
SYMBOL() 236
TEXT() 235
THICKNESS() 236
TITLE() 236
VIEW() 236
Графический интерфейс пользователя:
контекстная панель инструментов 23
контекстное меню 24, 42
основная панель инструментов 23
основное меню 22
палитры 32
рабочая область 24
стандартное меню рабочего листа 47
строка состояния 24
типы основного меню 45
Д
Двумерная графика:
contourplot() 230
coordplot() 226
densityplot() 229
display() 241
fieldplot() 231
gradplot() 231
imlicitplot() 227
inequal() 228
loglogplot() 228
logplot() 228
odeplot() 232
plot() 214, 235
PLOT-структура 233
plygonplot() 229
polarplot() 224
semilogplot() 228
textplot() 232
опции 214
пакет plots 224
пакет plottools 237
Дифференцирование и интегрирование:
D() 150
diff() 145
evalf() 147
int() 146
К
Команды:

add()114
algsubs() 126
coeff() 118
convert() 127
denom() 119
evalb() 285
has() 123
hastype() 124
is() 286
isolate() 151
lhs() 119, 284
limit() 397
map() 113, 122, 296
map2() 113
member() 110
mul() 114
nops() 120
numer() 119
op() 116, 120
piecewise() 242
protect() 279
remove() 115, 122, 296
rhs() 119, 284
select() 115, 122, 296
selectremove() 115, 296
seq() 107
series() 442
simplify() 126
sort() 117
specfunc() 124
subs() 125
subsop() 126
time() 310
type() 282
unapply() 140
unprotect() 278
whattype() 120
zip() 116, 296
разделитель 71
Константы 78
М
Модули:
module() 331
инкапсуляция 331
операция экспортирования, пакеты 331
параметр thismodule 335
реализация объектов 331
О
Ограничения на переменные:
about() 105
additionally() 103
assume() 101
coulditbe() 104
is() 104
Операнды выражения 120
Операторы:
break 295
error 303, 317
finally 319
next 295
return 316
try-catch 318
ветвления, if 290
присваивания 290
цикла, for-from 292
Операции:
$ (знак доллара) 108
if 292
intersect 110
minus 110
union 110
без операндов, %, %%, %%% 111
бинарные 276
диапазон 109
композиция двух функций, @ 277
логические 286
нейтральные 278
отношения 284
повторная композиция, @@ 277
присваивания 83
проверка типа, :: 302
унарные 275
П
Пакеты:
dotprod() 370

geom3d 380, 382
geometry 373
linalg 160
LinearAlgebra 160
powseries 445
Переменная 81
неизвестная 82
Полином 92
Преобразование выражений:
collect() 97
combine() 95
expand() 91
factor() 92
normal() 94
rationalize() 99
simplify() 87
Проверка решений:
eval() 138
map() 139
subs() 139
Пространственная графика:
contorplot3d() 260
coordplot3d() 257
cylinderplot() 255
display() 266
fieldplot3d() 261
gradplot3d() 261
implicitplot3d() 259
plot3d() 246, 262
PLOT3D-структуры 262
polygonplot3d() 261
spacecurve() 258
sphereplot() 257
textplot3d() 260
tubeplot() 259
опции 246
пакет plots 255
пакет plottools 264
Процедуры 299
возврат невычисленного значения 320
возвращаемое значение 299, 313
вычисление локальных переменных 305
локальные и глобальные переменные 304
массив args 302
неименованные 300
операнды типа 321
опции 308
передача параметров 301
сохранение в файле 323
таблица значений 309

Р

Рабочий лист 24
группа вычислений 25
область ввода 25
область ввода графики 25
область вывода 24, 29
секция 59
стандартное меню 47
форматы области вывода 29
форматы сохранения 41, 48
Решение дифференциальных уравнений:
D(), оператор 150
dsolve() 149
Решение уравнений:
fsolve() 141
isolve() 143
msolve() 143
rsolve() 143
solve() 135

С

Сложные типы данных:
массив 110
множество 108
неравенство 135
последовательность 106
список 108
таблица 112
уравнение 118, 134
Строки 79

Т

Типы данных 281
series 443

У

Устройство отображения графики
233
plotsetup() 234

Ч

Числа:
комплексные числа 77
обыкновенные дроби 74
радикалы 75
с плавающей точкой 76
целые 72

Ш

Шаблоны-заполнители 33

Э

Электронная таблица 35
влияющая ячейка 39
зависимая ячейка 39

Я

Язык Maple:
алгебраический контекст 284
алфавит 274
булевый контекст 285
выражение 281
дерево выражения 282
индексные имена 279
ключевые слова 275
лексемы 275
натуральные числа 280
символьные имена 278
строка 280
структурный тип 287
целые числа 280

Введение

Вам необходимо быстро, качественно и без ошибок решить математическую задачу в аналитическом виде (конечно, в случае если это вообще возможно) — следует обратиться к программе Maple 6. Если решение задачи возможно только в численном виде — Maple 6 поможет и в этом случае. Вы привыкли производить анализ данных в какой-либо системе электронных таблиц, например, MS Excel, — Maple 6 предложит вам электронную таблицу с привычным графическим интерфейсом, предоставив в распоряжение огромный арсенал разнообразных математических и статистических функций для обработки табличных данных. Надо быстро построить график сложной функции или трехмерной поверхности — что ж, Maple 6 и здесь не заменим. "Да что же это такое! — воскликнет иной читатель. — Неужели Maple 6 может делать практически все, что необходимо при работе с математическими моделями?" И он будет не далек от истины.

Новая версия Maple 6 системы аналитических вычислений, или системы компьютерной алгебры, канадского университета Waterloo, появившаяся в конце декабря 2000 года, позволяет делать не только то, что перечислено, но и многое другое. Интуитивно-ясный и простой язык Maple позволяет программировать решения задач, если сам Maple 6 не предлагает ее решения в виде встроенной или библиотечной функции (команды), даже людям далеким от программирования и не являющимися специалистами в области программирования и информатики.

Maple могут использовать (и используют) для решения своих задач не только специалисты в области математики, но и специалисты в других прикладных областях. Главное, чтобы они понимали и имели представление о тех математических моделях, которые применяют в своих исследованиях.

Достаточно широко Maple применяется в образовании, причем не только математических дисциплин, но и технических дисциплин. Например, его можно с успехом использовать для преподавания разнообразных направлений механики: начиная с классической теоретической механики, сопротивления материалов, строительной механики и т. д. и заканчивая квантовой

механикой. Возможности Maple в области разработки анимационной графики позволяют наглядно показать обучающимся те или иные законы движения или взаимодействия тел.

Для студентов Maple является неоценимым помощником в изучении разнообразных математических методов, освобождая их от рутинных математических вычислений и сосредотачивая их внимание на существе изучаемого метода. В поставку системы аналитических вычислений Maple входит специальный пакет `student`, содержащий большой набор функций для выполнения именно студентами разнообразных математических преобразований: взятие интеграла по частям, замена переменных в неопределенном и определенном интегралах, отыскание максимума и минимума функций и т. д.

Maple 6 реализован на разных платформах, среди которых Windows, VMS, UNIX и Linux. Везде он предлагает пользователю интерфейс, к которому он привык при работе в соответствующей операционной системе. Наша книга посвящена реализации этой мощной системы аналитических (и не только) вычислений в среде Windows 95/98/NT, хотя следует сказать, что язык Maple, на котором пользователь общается с программой, един для всех платформ, поэтому нашу книгу могут читать не только пользователи упомянутого семейства операционных систем Windows. Для тех, кто использует Maple 6 в системе Windows, работа осуществляется через ясный и понятный графический интерфейс, стандартный для любого приложения, разработанного для этой операционной системы, например, семейства приложений Microsoft Office. Вся работа осуществляется на рабочем листе, который можно рассматривать как "документ" в терминологии упомянутых приложений MS Office. Пользователь в интерактивном режиме на рабочем листе после приглашения Maple (символ ">") вводит необходимые команды, и по завершении ввода (при нажатии клавиши <Enter>) интерпретатор Maple тут же выдает результат выполнения введенной команды. Таким образом, вся работа осуществляется в интерактивном режиме.

Для того чтобы читатель оценил ту простоту, с которой решаются задачи в Maple, мы сочли уместным привести во введении несколько примеров решения задач в этой действительно великолепной и простой системе аналитических вычислений.

Пример 1: неопределенный интеграл

Предположим, что нам надо вычислить неопределенный интеграл от некоторой функции. Для этого, прежде всего, следует на рабочем листе Maple определить саму функцию. Это осуществляется с помощью следующей операции присваивания, которая набирается пользователем на рабочем листе сразу же после приглашения Maple:

```
> f := x -> ln(x)/(x^3);
```

Нажав клавишу <Enter>, мы тем самым дадим системе знак, что введенный нами оператор следует выполнить. Результатом его выполнения будет создание функции `f(x)`:

$$f := x \to \frac{\ln(x)}{x^3}$$

К созданной функции теперь можно обращаться так же, как это обычно принято в математике. Например, чтобы получить значение функции `f(x)` в точке `1`, достаточно набрать `f(1)` и т. д.

Обратим внимание, что вывод Maple осуществляется в принятой математической нотации.

Итак, функция задана, теперь следует вычислить от нее неопределенный интеграл. Нет ничего проще — набираем в следующей строке команду вычисления интеграла, нажимаем <Enter>, и Maple печатает нам ответ:

```
> int(f(x),x);
```

$$-\frac{1}{2}\frac{\ln(x)}{x^2}-\frac{1}{4}\frac{1}{x^2}$$

Как видим, название команды для вычисления интеграла `int()` интуитивно ясно (хотя следует заметить, что эта ясность очевидна для тех, кто знает английский язык, но мы убеждены, что современный читатель владеет им в достаточной степени), ее параметрами являются сама функция `f(x)` и переменная `x`, по которой осуществляется интегрирование. Полученное значение неопределенного интеграла Maple печатает в привычной математической нотации. Правда, для полноты результата следует к вычисленному Maple значению интеграла добавить произвольную константу `C`.

Замечание

Полученный результат можно выделить на рабочем листе, скопировать в Буфер обмена и вставить в виде рисунка в документ MS Word.

Внимание!

Система Maple при экспортировании своего вывода в виде математической нотации в приложение MS Word не формирует, как может показаться, привычный для пользователей этого популярного текстового процессора объект редактора формул Microsoft Equation 3.0. Результатом подобного экспортирования будет графический объект, который в документе Word можно редактировать встроенным графическим редактором.

Итак, неопределенный интеграл вычислен практически моментально, но студент какого-нибудь технического университета скажет: "А где промежуточные выкладки, как я смогу показать такое "решение" преподавателю?" Не стоит отчаиваться и тут же отказываться от применения Maple в своей

повседневной учебе. Получить промежуточные выкладки нам поможет пакет student, который следует подключить специальной командой with(), чтобы получить доступ к включенной в него функции интегрирования по частям неопределенного интеграла intparts(), которая реализует известную формулу: $\int u\,dv = u\,v - \int v\,du$. Для правильной работы этой команды ей следует передать два параметра: первый — интеграл от функции, а второй — явный вид функции *u* из приведенной формулы. Если, наверное, с заданием явного вида функции все ясно, то вот как передать в нее интеграл? Для этого в Maple для ряда команд существует так называемая отложенная форма, которая позволяет задать, например, интеграл, но не вычислять его. Такие команды имеют аналогичный вычисляемым командам синтаксис, но их названия начинаются с прописной буквы:

```
> Int(f(x),x);
```

$$\int \frac{\ln(x)}{x^3}\,dx$$

Именно этим средством Maple мы и воспользуемся для вычисления формулы интегрирования по частям:

```
> with(student):
> intparts(Int(f(x),x),ln(x));
```

$$-\frac{1}{2}\frac{\ln(x)}{x^2} - \int -\frac{1}{2}\frac{1}{x^3}\,dx$$

Ответ практически готов, так как интеграл в полученном выражении является табличным, и его вычисление не представляет никакого труда. Обратите внимание, что при подключении пакета student командой with() мы завершили ее двоеточием (:), а не точкой с запятой (;), как все остальные команды. Дело в том, что любую команду Maple можно завершить одним из этих символов, но в первом случае Maple не отображает результатов выполнения команды, а во втором осуществляет вывод в строке рабочего листа, следующей за введенной командой. В нашем случае с командой with() мы таким способом предотвратили вывод на рабочий лист списка всех команд, включенных в пакет student.

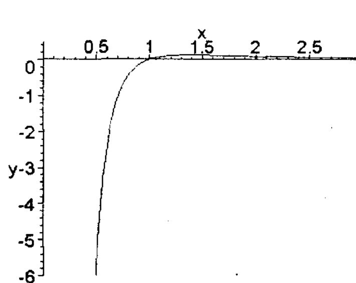
Как видно из этого примера, Maple позволяет не только получить окончательный ответ с помощью обращения к одной команде, но и воспользоваться формулой интегрирования по частям при вычислении неопределенного интеграла, а последнее предполагает, конечно, ее знание.

Пример 2: график функции и поверхности

Системы аналитических вычислений, и Maple здесь не исключение, привлекают многих своих пользователей удобными и хорошо реализованными возможностями отображения графической информации. Ориентированные на выполнение математических преобразований в аналитической форме, они предоставляют инструменты для работы с разнообразными геометрическими объектами. Простейшим из них является график заданной в явном виде функции одной независимой переменной.

Отобразить, или начертить график функции одной переменной в Maple можно простым обращением к команде `plot()`, обязательными параметрами которой являются сама функция и независимая переменная с заданным интервалом изменения, на котором и вычерчивается график. Для задания интервала изменения переменной в Maple предусмотрен оператор "диапазон" `a..b`, в котором действительная величина `a` должна быть меньше `b`. Необязательный третий параметр ограничивает отображаемую область значений функции, что порой является полезным при работе с разрывными функциями. Именно для этих целей в следующей команде отображения графика функции `f(x)`, созданной нами в примере 1, и используется третий параметр.

```
> plot(f(x),x=0..3,-6..0.5,);
```



Замечание

Кроме упомянутых выше трех параметров функция `plot()` имеет большое количество опций, задание которых влияет на оформление получаемого графического образа: толщина и цвет линии графика и осей, наличие надписи на графике, тип осей и т. п. Правда, следует отметить, что значения всех опций можно установить на контекстной панели инструментов, отображаемой при выделении графика на рабочем листе, или в контекстном меню, которое вызывается, если нажать правую кнопку мыши в области рисунка, но об этом мы расскажем в первой главе книги.

На самом деле, графический интерфейс пользователя системы Maple позволяет выполнить практически все необходимые, наиболее употребительные

команды, не прибегая к вводу с клавиатуры соответствующих команд. Если подвести указатель мыши к объекту в области вывода Maple и щелкнуть правой кнопкой мыши, то в зависимости от типа содержащегося в ней объекта (графика, выражения) отобразится соответствующее контекстное меню со списком применимых к данному объекту команд Maple. Например, наберем в строке ввода рабочего листа после его стандартного приглашения следующую команду (функция `f(x)`, определенная нами в примере 1):

```
> f(x);
```

Maple вычислит его и отобразит явный вид определенной нами функции

$$\frac{\ln(x)}{x^3}$$

Теперь наведем указатель мыши на этот вывод системы Maple, щелкнем правой кнопкой мыши, выберем из контекстного меню команду **Plots ➢ 2-D Plot**, и Maple выполнит команду `smartplot()` (аналог команды `plot()` с заранее определенными значениями параметров), отображающую график функции, представленной в области вывода.

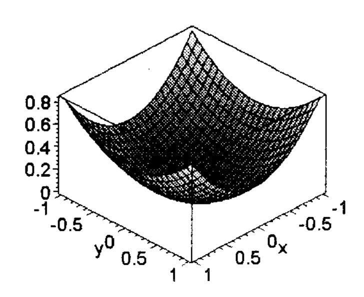
Внимание!

Следует сказать, что если читатель проделает указанные действия, то увидит вместо графика функции, полученного нами ранее с помощью функции `plot()`, всего лишь оси координат. Это связано с тем, что наша функция имеет в точке $x=0$ разрыв (стремится к $-\infty$) и определена только при положительных значениях независимой переменной, а функция `smartplot()` по умолчанию строит график в диапазоне изменения независимой переменной от `-10` до `+10` и не ограничивает область значений функции. Чтобы увидеть уже построенный нами график, следует отобразить для полученного графика контекстное меню (нажать правую кнопку мыши при наведенном на график ее указателе) и в диалоговом окне **Axis Ranges**, отображаемом командой **Axes ➢ Ranges**, установить соответствующие диапазоны для независимой переменной и области вывода.

Maple позволяет отображать графики функций, определенных неявным образом, рисовать параметрически заданную кривую, а также создавать и отображать стандартные плоские графические объекты: точки, многоугольники, окружности и т. д., но все это можно сделать, подключив пакет `plots`, который содержит все необходимые команды.

Графическое отображение поверхности, заданной в явном виде, отличается от графического отображения функции одной переменной только названием команды (`plot3d`) и тем, что для ее правильной работы необходимо задать, кроме уравнения отображаемой поверхности, диапазон изменения двух ее независимых переменных:

```
> plot3d(x^2/2+y^2/3, x=-1..1, y=-1..1);
```



Однако возможности редактирования трехмерных изображений намного богаче, чем при работе с плоскими двумерными образами. Можно посмотреть на полученную трехмерную поверхность с разных точек зрения, поворачивая ее в пространстве. Для этого достаточно "прихватить" поверхность мышью (подвести курсор мыши в область отображения, нажать левую кнопку и не отпускать ее), и перемещая мышь, поворачивать поверхность в том или ином направлении. Можно раскрасить ее, используя предлагаемые цветовые схемы или с помощью собственной схемы, добавить разнообразную световую подсветку, отобразить разные типы или вообще убрать оси координат и многое другое.

Надеемся, эти два совсем небольших примера в какой-то мере убедили читателя, не знакомого с предыдущими версиями системы Maple, в том, что это действительно простая, удобная и мощная система компьютерной алгебры, на изучение которой стоит потратить некоторое время.

Для тех, кто работал и знает возможности предыдущих версий Maple V Release 4 и 5, конечно, наибольший интерес представляют новые возможности, предлагаемые новой версией системы Maple 6. Следующий раздел нашего, возможно затянувшегося, введения именно и посвящен тем новым возможностям, которые предлагает новая версия системы аналитических вычислений Maple 6. Следует заметить, что новичку тоже не стоит пропускать этот раздел, так как содержащаяся в нем информация, возможно, еще более убедит его в том, что система Maple 6 — стоящая система, значительно увеличивающая его возможности в решении математических задач.

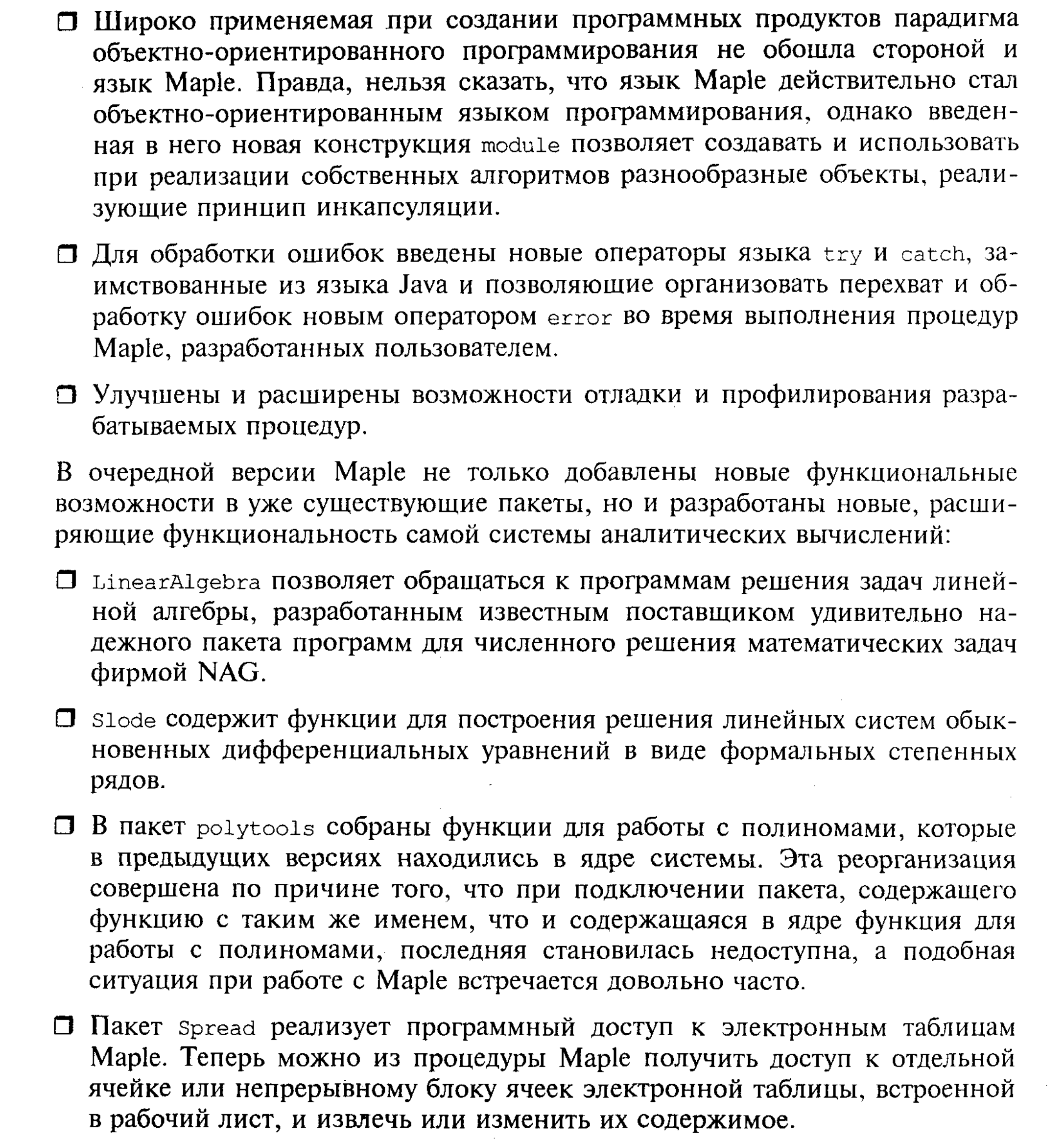
Новое в Maple 6

Среди разработчиков программного обеспечения принято присваивать новой версии продукта следующий номер, если эта новая версия существенно отличается по своим функциональным возможностям от предыдущей. В Maple 6 введено много новшеств и улучшений, и сразу же следует обратить внимание на три новые возможности:

- Новый пакет `LinearAlgebra` позволяет пользователю системы обращаться к откомпилированному коду программ линейной алгебры из известного среди специалистов по численным методам пакета NAG (North Algorithmic Group), причем с возможностью задания произвольного числа значащих цифр в мантиссе представления чисел с плавающей точкой. Возможность обращения к откомпилированным программам численного решения разнообразных математических задач предоставляют многие системы компьютерной алгебры, но возможность при этом использовать в представлении чисел мантиссу практически неограниченной длины — это большое достижение разработчиков системы Maple 6. Подобное решение практически снимает проблему вычислительной неустойчивости алгоритмов, известную любому специалисту по численным методам. Реализация этой возможности осуществлена с помощью специально разработанной технологии вызова внешних процедур, написанных на компилируемом языке, например, C++ или Fortran. Причем следует отметить, что эта технология доступна и для пользователей системы Maple 6, т. е. теперь для ускорения численных расчетов или реализации отсутствующего в Maple 6 численного алгоритма можно воспользоваться одним из упомянутых языков программирования, создать на нем требуемое решение и обращаться к нему непосредственно из рабочего листа Maple.
- Использование практически всей мощи Maple 6 из такого популярного приложения обработки данных в виде электронных таблиц, как Microsoft Excel 2000 из семейства приложений MS Office 2000, с помощью разработанной для этого приложения специальной надстройки. Всем пользователям Excel известны его неудобства при работе с функциями — с подключением надстройки Maple для построения графика функции не надо будет формировать таблицу ее значений, а достаточно будет прямо в ячейке рабочего листа Excel написать вызов соответствующей команды Maple из графического пакета.
- Пользователям предыдущих версий Maple известна проблема переноса содержимого рабочего листа Maple в наиболее популярной у нас в стране текстовый процессор MS Word: весь вывод результатов выполнения команд в форме общепринятой математической нотации превращался при переносе содержимого рабочего листа в обычные команды Maple. В версии Maple 6 добавлена возможность сохранения рабочих листов в формате RTF (Rich Text Format), поддерживаемом Word. Содержащиеся в предыдущей версии форматы сохранения LaTeX и HTML, естественно, продолжают поддерживаться и в новом Maple.

Кроме этих трех наиболее важных, не только с нашей точки зрения, но и с точки зрения самих разработчиков системы, добавлений следует отметить существенное расширение возможностей программирования в Maple.

- Об одной из новых возможностей мы уже упоминали — это вызов внешних программ, написанных на языке С.



- Широко применяемая при создании программных продуктов парадигма объектно-ориентированного программирования не обошла стороной и язык Maple. Правда, нельзя сказать, что язык Maple действительно стал объектно-ориентированным языком программирования, однако введенная в него новая конструкция `module` позволяет создавать и использовать при реализации собственных алгоритмов разнообразные объекты, реализующие принцип инкапсуляции.
- Для обработки ошибок введены новые операторы языка `try` и `catch`, заимствованные из языка Java и позволяющие организовать перехват и обработку ошибок новым оператором `error` во время выполнения процедур Maple, разработанных пользователем.
- Улучшены и расширены возможности отладки и профилирования разрабатываемых процедур.

В очередной версии Maple не только добавлены новые функциональные возможности в уже существующие пакеты, но и разработаны новые, расширяющие функциональность самой системы аналитических вычислений:

- `LinearAlgebra` позволяет обращаться к программам решения задач линейной алгебры, разработанным известным поставщиком удивительно надежного пакета программ для численного решения математических задач фирмой NAG.
- `Slode` содержит функции для построения решения линейных систем обыкновенных дифференциальных уравнений в виде формальных степенных рядов.
- В пакет `polytools` собраны функции для работы с полиномами, которые в предыдущих версиях находились в ядре системы. Эта реорганизация совершена по причине того, что при подключении пакета, содержащего функцию с таким же именем, что и содержащаяся в ядре функция для работы с полиномами, последняя становилась недоступна, а подобная ситуация при работе с Maple встречается довольно часто.
- Пакет `Spread` реализует программный доступ к электронным таблицам Maple. Теперь можно из процедуры Maple получить доступ к отдельной ячейке или непрерывному блоку ячеек электронной таблицы, встроенной в рабочий лист, и извлечь или изменить их содержимое.

Кроме перечисленных новшеств, в новую версию Maple 6 включены и другие изменения и улучшения, о которых мы будем постоянно информировать читателя по мере изложения материала книги. Завершая краткое описание новых возможностей, следует напомнить читателю, что, как в любой версии Maple, он может познакомиться с ними, обратившись к странице справки "New Features That have Been Added to Maple 6" ("Новые возможности, добавленные в Maple 6"), доступной по команде **Help > What's New**.

Структура книги

Книга состоит из трех частей, разделенных на главы, которые, в свою очередь, составлены из разделов. Нумерация глав сквозная, тогда как разделы, рисунки и примеры имеют самостоятельную нумерацию в пределах каждой главы с указанием ее номера, например, рис. 5.2 — это рисунок 2 главы 5.

Часть I посвящена последовательному введению в технологию работы с программой аналитических вычислений Maple 6.

Глава 1 посвящена вопросам взаимодействия пользователя с программой через ее графический интерфейс. Подробно описываются его основные элементы: меню, панели инструментов и рабочие листы, а также демонстрируется техника их использования.

В главе 2 вводятся основные объекты языка Maple и команды для решения наиболее часто возникающих математических задач: решение систем линейных и нелинейных уравнений, дифференцирование и интегрирование функций, преобразование математических выражений, решение обыкновенных дифференциальных уравнений. Также кратко рассматривается структура внутреннего представления в Maple аналитических выражений и основные команды, работающие с ней.

В главе 3 основное внимание уделено организации Maple и существующим пакетам, в которых сгруппированы команды для решения задач определенного класса. Рассмотрен новый пакет линейной алгебры `LinearAlgebra`, пакет `student`, а также пакеты, предназначенные для решения обыкновенных дифференциальных уравнений и уравнений в частных производных.

Глава 4 полностью посвящена графическим возможностям Maple. Подробно рассмотрены возможности двумерной и трехмерной графики, а также создание двумерных и трехмерных анимаций.

В главе 5 излагаются основы программирования в Maple, описываются все конструкции языка Maple, дается введение в программирование модулей и работе с внешними файлами. Особое внимание уделено новой возможности перехвата и обработки исключительных состояний во время выполнения программы Maple.

Глава 6 описывает работу с надстройкой Maple для программы MS Excel, расширяющей функциональность этого популярного приложения для работы с табличными данными.

Часть II книги содержит многочисленные примеры решения задач высшей математики из разделов, изучаемых в программе технических университетов: аналитическая геометрия, алгебра, дифференцирование и интегрирование функций, дифференциальные уравнения и ряды, дискретная математика.

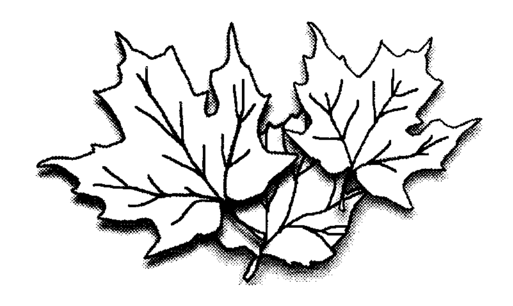
В заключительной части III показывается применение Maple для решения задач теоретической механики, строительной механики и теории упругости.

Используемые обозначения

На протяжении всей книги используются следующие обозначения:

➢	Используется для задания команд меню и подменю. Например, вызов команды **RTF**, сохраняющей рабочий лист в формате RTF и расположенной в подменю **Export As** меню **File**, в тексте книги выглядит так: **File ➢ Export As ➢ RTF**
File	Полужирный шрифт применяется к названиям меню, подменю и команд, равно как и к наименованиям диалоговых окон и их элементов управления.
`> nops(f);`	Моноширинный шрифт используется для представления содержимого области ввода рабочего листа Maple.
$\operatorname{cosec}(x)$	Шрифт математической нотации Maple. Применяется для представления области вывода рабочего листа Maple и для обозначения математических формул в тексте.

Часть I



Основы Maple

Глава 1. Графический интерфейс пользователя

Глава 2. Основные объекты и команды

Глава 3. Пакеты

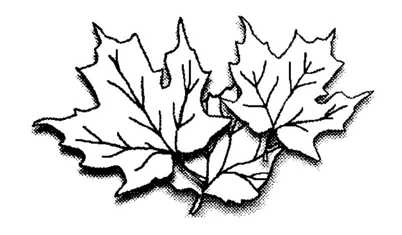
Глава 4. Графика

Глава 5. Основы программирования в Maple

Глава 6. Maple в Excel

В первой части книги вы познакомитесь с основными возможностями системы Maple. Подробно описываются элементы интерфейса: меню, панели инструментов и рабочие листы, а также наиболее часто используемые объекты языка Maple и команды для решения распространенных математических задач. Кратко рассматривается представление в Maple аналитических выражений и основные команды по их преобразованию. Кроме того, в данной части обсуждаются графические возможности системы, даны основы программирования, приведены способы доступа к функциям Maple из рабочего листа MS Excel.

ГЛАВА 1



Графический интерфейс пользователя

Пользователь взаимодействует с любым программным приложением через его интерфейс, который может быть представлен единственной строкой ввода необходимых текстовых команд *(интерфейс командной строки)*, всей площадью экрана монитора, в разных частях которого следует вводить для работы приложения текстовую информацию *(экранный интерфейс)*, причем, если используется графический режим работы монитора, то подобный интерфейс называется *графическим интерфейсом*. Приложения, разработанные для операционной системы Windows, практически все предоставляют пользователю именно последний тип интерфейса, который представляет собой "окно" (по принятой в Windows терминологии) с расположенными в нем разнообразными стандартными компонентами и элементами управления: меню, панели инструментов, поля ввода различных типов и т. д.

Для того чтобы успешно использовать любую программу (приложение), следует достаточно основательно познакомиться с ее интерфейсом и изучить большинство его возможностей, ведь именно через него мы заставляем наше приложение выполнить то, что нам требуется. Эта глава посвящена описанию основных компонентов интерфейса пользователя системы Maple, которому мы уделим столько времени, сколько необходимо для того, чтобы начать работу. В дальнейшем, по мере необходимости, мы будем снова обращаться к возможностям, предоставляемым интерфейсом. Наша основная цель — научить решать задачи с помощью Maple, а умение работать с интерфейсом придет по мере накопления опыта работы с системой.

1.1. Общий вид

Приложение Maple 6 имеет стандартный графический интерфейс пользователя для программ, работающих под управлением операционных систем семейства Windows. При загрузке программы на экране монитора отображает-

ся окно интерфейса, представленное на рис. 1.1. В нем можно выделить несколько основных компонентов (частей).

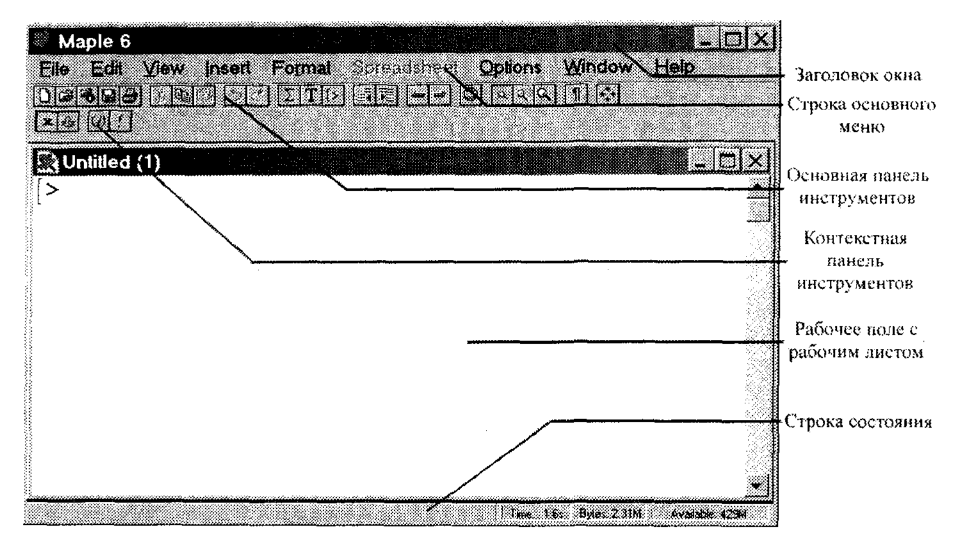


Рис. 1.1. Общий вид интерфейса приложения Maple 6

В верхней части, непосредственно под заголовком окна, находится строка *главного*, или *основного*, меню, содержащая список основных меню системы Maple. При нажатии левой кнопкой мыши на заголовке какого-нибудь из них, например, на **File**, выпадает соответствующее меню, и в нем можно найти команды и заголовки подменю. В дальнейшем мы не будем явно указывать, что нажата левая кнопка мыши. Выражение "нажать кнопку мыши" будет соответствовать "нажать левую кнопку мыши". Там, где необходимо использовать правую кнопку мыши (например, для вызова контекстного меню), это будет указываться явным образом.

Следует отметить особенность главного меню приложения Maple — зависимость от контекста: если на рабочем листе выделен графический объект, то главное меню изменяется, отражая специфику работы с графикой Maple. На рис. 1.1 показана стандартная строка меню Maple, когда курсор находится в области ввода команд или в области вывода, в которой отображается результат выполнения математических команд. На рис. 1.2 можно посмотреть, как будет выглядеть основное меню, если на рабочем листе выделен вывод графической команды отображения трехмерной поверхности.

Еще одна особенность основного меню связана с тем, что некоторые его меню не доступны, пока на рабочем листе не появится соответствующий объект, для работы с которым они предназначены. На рис. 1.1 — это меню **Spreadsheet**, а на рис. 1.2 — меню **Animation**. Первое предназначено для работы со встраиваемыми электронными таблицами Maple, а второе — для

управления анимационной графикой, позволяющей отобразить зависимость графического вывода от одного или нескольких параметров.

Замечание

Об электронных таблицах Maple, которые не следует путать с электронными таблицами MS Excel, речь пойдет немного позже в этой же главе, тогда как создание анимационной графики освещается в гл. 4.

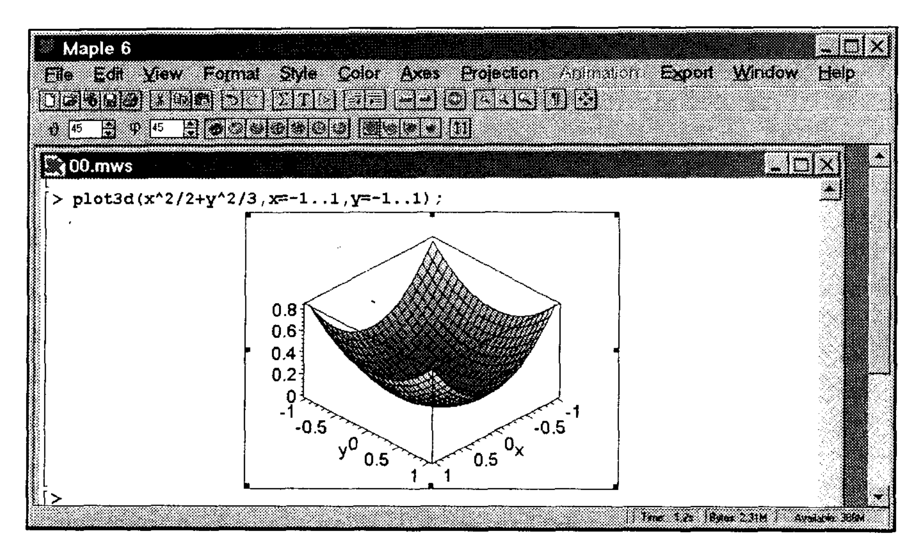


Рис. 1.2. Вид основного меню для работы с графикой, расположенной на рабочем листе

Ниже строки главного меню расположена *основная панель инструментов* с рядом кнопок, дублирующих наиболее часто используемые команды основного меню. Щелчок мыши на кнопке приводит к выполнению ассоциированной с ней команды. Эта панель не имеет свойство адаптироваться к выделенным на рабочем листе объектам, состав ее кнопок постоянен и не меняется при перемещении курсора по рабочему листу. Она полностью меняется только при вызове справочной системы Maple по командам меню **Help**.

Замечание

Со справочной системой Maple, которая несколько отличается от привычных для большинства пользователей Windows-приложений справочных систем, мы познакомимся немного далее в этой же главе.

Непосредственно под основной панелью инструментов расположена *контекстная панель инструментов*, вид которой зависит от того, в какой области рабочего листа расположен курсор и что в этой области отображается. Существует пять видов контекстных панелей инструментов: для выделен-

ного двумерного графика, для выделенного трехмерного графика, для выделенной анимационной графики, для области вывода и для области ввода рабочего листа, причем в последнем случае вид контекстного меню различен при использовании стандартной математической записи команд Maple или записи команд в стандартной нотации Maple. На рис. 1.1 показана контекстная панель инструментов области ввода рабочего листа при использовании стандартной нотации Maple для записи команд.

Большую часть окна интерфейса занимает *рабочая область*. Именно в ней располагаются рабочие листы, в которых вводятся команды и отображаются результаты их выполнения. Интерфейс приложения Maple является многодокументным, позволяющим открыть и работать одновременно с несколькими рабочими листами, которые и являются "документами" Maple. Более того, приложение Maple позволяет связывать рабочие листы посредством включения в них *гиперссылок* на другие рабочие листы или файлы справки, но об этом мы поговорим подробнее в разделе этой главы, посвященном вопросам документирования сеанса работы в Maple.

В нижней части интерфейса расположена *строка состояния*, в которой отображаются некоторые параметры, относящиеся к исполняющей системе Maple, а также краткая информация относительно выбранной команды меню или кнопки панели инструментов.

Во время работы для всех объектов рабочего листа (графика, вывод результатов выполнения команд, команда в области ввода) можно отобразить *контекстное меню*, содержащее набор команд, применимых к данным объектам. Для этого достаточно расположить указатель мыши над соответствующим объектом и нажать правую кнопку мыши. Рядом с указателем мыши появится контекстно-зависимое меню — небольшая панель с набором применимых к данному объекту наиболее употребительных команд.

1.2. Рабочие листы

Технология работы в Maple представляет собой интерактивный сеанс: пользователь вводит на рабочем листе команды и нажатием клавиши <Enter> передает их на выполнение исполняющей системе (ядру) Maple. Все вводимые команды и отображаемые результаты вычислений представляют собой содержимое *рабочего листа* — основного документа, который Maple создает и с которым он работает. При завершении сеанса работы его можно сохранить на диске в файлах разных форматов, а при очередном сеансе открыть и снова выполнить все команды, содержащиеся в нем, или произвести его корректировку. На рис. 1.3 показан рабочий лист с командами и результатами их выполнения в окне интерфейса пользователя системы аналитических вычислений Maple.

Рабочий лист состоит из *области ввода* и *области вывода*. В первой пользователь вводит команды Maple, которые передаются на выполнение ядру сис-

темы. В области вывода отображаются результаты выполнения команд и операторов языка Maple, а также двумерная и трехмерная графика, создаваемая графическими командами Maple, если задан режим вставки графики в рабочий лист, а не отображения ее в отдельном окне. В случае отображения графики на рабочем листе область вывода будем называть *областью вывода графики*. Все указанные области можно видеть на рис. 1.3.

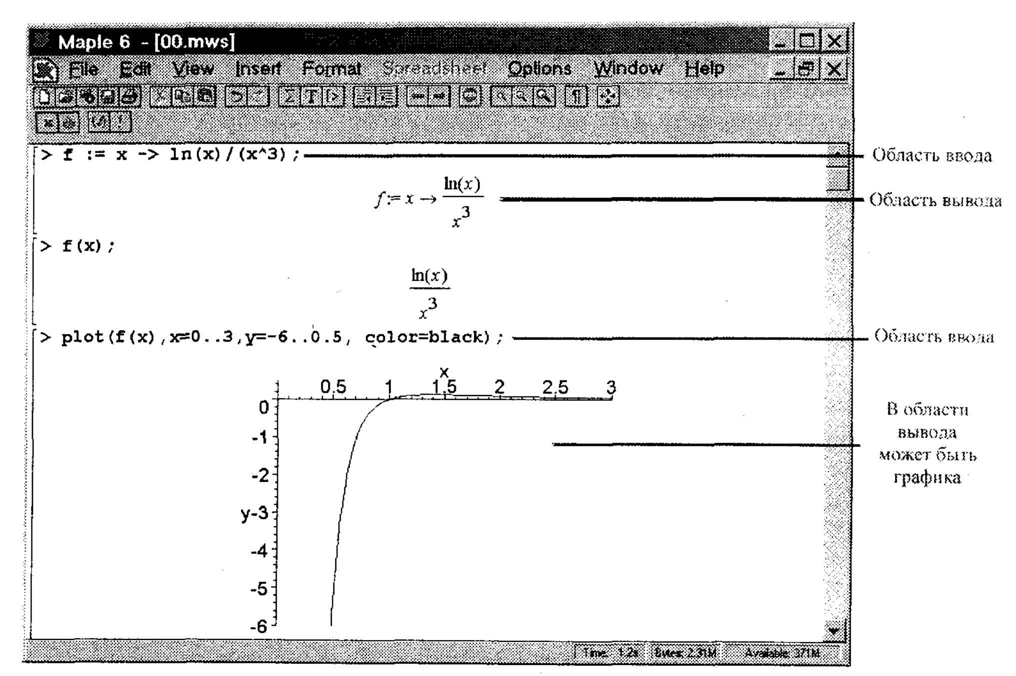


Рис. 1.3. Рабочий лист Maple

Содержимое областей ввода и вывода образуют *группу вычислений*, или просто *группу*, которая на рабочем листе отмечается слева квадратной скобкой. Группа вычислений может содержать несколько областей ввода и, соответственно, вывода: Основное свойство группы заключается в том, что все ее операторы и команды выполняются за одно обращение к исполняющей системе Maple, т. е. нажатие клавиши <Enter> приводит к вычислению всех ее операторов и команд.

1.2.1. Область ввода

Область ввода — это область рабочего листа, в которой пользователь вводит информацию. Эта информация может быть двоякого рода: команды и операторы Maple или текстовые комментарии.

По умолчанию при создании нового рабочего листа (при загрузке Maple новый рабочий лист создается автоматически) устанавливается режим ввода

команд и операторов. Указанием на это является приглашение ввода в строке рабочего листа — символ >, сразу же после которого расположен мерцающий курсор. Пользователь вводит с клавиатуры команды и нажатием клавиши <Enter> передает их на обработку символьному анализатору Maple, который в зависимости от того, правильно или нет они набраны, отображает в поле вывода либо результат выполнения команды, либо сообщение об ошибке.

Команды можно отображать либо в форме синтаксиса языка Maple, либо в виде привычной математической записи. Например, для вычисления интеграла от функции f(x) можно в поле ввода набрать команду `int(f(x),x)`, либо установить режим, при котором эта команда отобразится в привычной математической записи $\int f(x)\,dx$. Для этого следует до начала набора текста команды Maple выполнить команду **Insert ➤ Standart Math Input**, результатом выполнения которой будет смена курсора в области ввода на вопросительный знак. Также появится контекстная панель инструментов ввода команд Maple при их отображении в привычной всем математической форме записи (при условии, что вводимая команда имеет соответствующую математическую запись, например, как в случае с вычислением интеграла). При вводе команды с клавиатуры в поле ввода контекстной панели будет отображаться именно то, что вводится, а в области ввода эта же команда будет отображена в математической нотации по завершении ввода нажатием клавиши <Enter>, как показано на рис. 1.4.

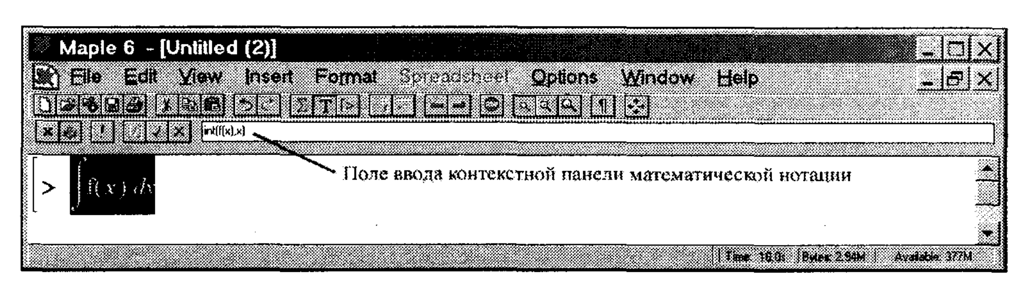


Рис. 1.4. Ввод и отображение команд в математической нотации

Замечание

На рис. 1.4 отображение математической нотации команды Maple нами выделено специально, чтобы показать читателю, что он будет видеть в поле ввода контекстной панели при вводе с клавиатуры команды вычисления интеграла. В действительности при ее вводе в области ввода будет отображаться вопросительный знак, пока не будет завершен ввод нажатием клавиши <Enter>.

Переключаться между режимами ввода команд в форме синтаксиса Maple и математической записи можно также с помощью кнопки ![кнопка], расположен-

ной на контекстной панели инструментов. Эта кнопка работает как обычный переключатель, что позволяет в любой момент заменить команды, записанные в форме синтаксиса Maple на их математическую нотацию и наоборот, предварительно установив на них курсор.

Режим ввода *текстовых комментариев* — это такой режим, при котором любая вводимая в области ввода информация рассматривается как обычный текст, а не как команды Maple, которые следует выполнить. Такие комментарии удобны, когда нужно разъяснить, что будет выполнять последующий оператор, или просто необходимо описать постановку задачи и основные моменты алгоритма ее решения. Следует отметить, что система Maple достаточно широко используется по всему миру при обучении математике и некоторым техническим дисциплинам. Поэтому возможность вставки в рабочий лист текстовых комментариев, позволяющих создать учебный материал для изучения и демонстрации применения каких-то понятий, методов, алгоритмов и т. п., является достаточно полезным средством.

Примеры команд в области ввода вместе с текстовыми комментариями представлены на рис. 1.5. При его внимательном рассмотрении видно, что в текстовый комментарий можно вставлять формулы, а также выделять фрагменты текста.

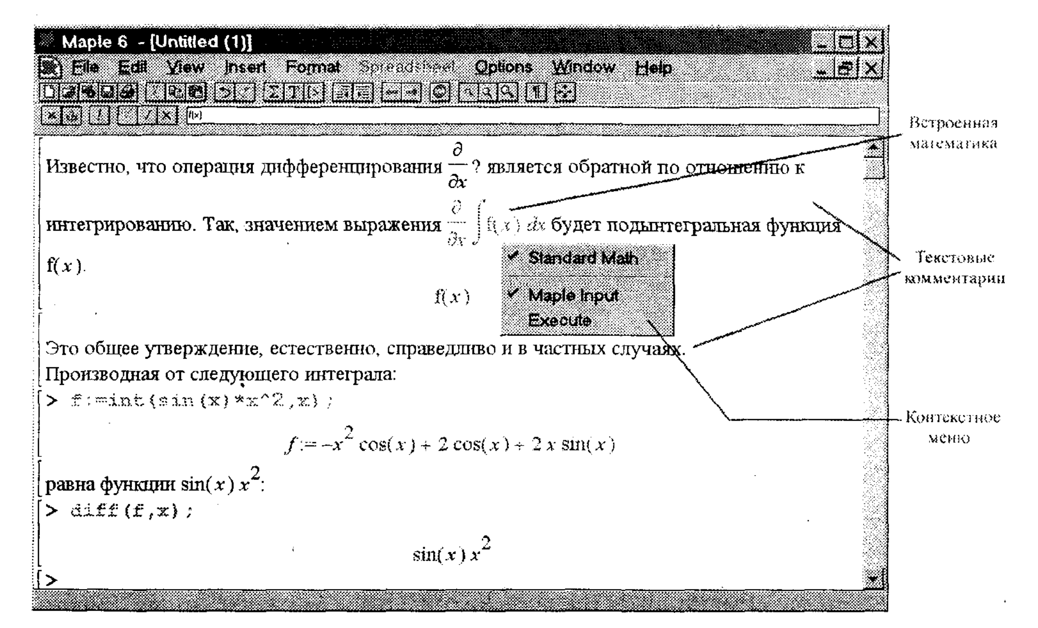


Рис. 1.5. Рабочий лист с текстовыми комментариями и командами Maple

Для того чтобы начать вставку текстового комментария в рабочий лист Maple, следует выполнить команду **Insert > Text** или нажать кнопку **T** на основ-

ной панели инструментов. Можно поместить в текстовый комментарий математическую формулу. Для этого достаточно выполнить команду **Insert ➢ Standard Math** или нажать кнопку на основной панели инструментов. Технология ввода формул в текстовый комментарий аналогична технологии ввода команд Maple в математической нотации.

Maple позволяет вставлять в комментарии выполняемые команды в форме собственного синтаксиса (**Insert ➢ Maple Input**) или в форме математической записи (**Insert ➢ Standart Math Input**). Отличие подобных вставок от описанных выше вставок математических формул заключается в том, что введенные таким способом формулы можно в любой момент вычислить с помощью команды **Execute** контекстного меню для вставленной формулы или нажатием кнопки на контекстной панели инструментов. На рис. 1.5 показано контекстное меню для вставленной в текстовый комментарий команды вычисления интеграла.

Замечание

Отличить встроенную в текстовый комментарий формулу от соответствующей выполнимой команды Maple можно также по цвету шрифта: первая обычно черная, как все символы текстового комментария, а вторая отображается красным цветом. Это связано с разными форматами шрифта для команд Maple и текстовых комментариев. По умолчанию в Maple для них установлены именно те цвета, которые мы только что упомянули, хотя в любой момент можно изменить форматирование ввода Maple, но об этом далее в этой же главе в разделе о возможностях документирования в Maple.

Вернемся к процедуре ввода команд. Если команда достаточно длинная и не помещается в одной строке рабочего поля, то Maple автоматически переходит на следующую строку. Введенная при этом длинная и расположенная в нескольких строках команда является одним целым. Пользователю нет необходимости следить за переносом длинных команд или больших комментариев.

Внимание!

Подобный перенос осуществляется только тогда, когда вводимая команда представлена не как поток символов, а разбита пробелами на слова. Если эти условия не выполнены, то при наборе длинных строк без пробела пользователь не видит набираемого текста, если его длина превысила ширину видимой части рабочего листа, хотя сам введенный текст присутствует на рабочем листе. В этом можно убедиться, развернув окно Maple на весь экран монитора, а рабочий лист на всю рабочую область.

В одной строке можно вводить несколько операторов, разделенных точкой с запятой (;) или двоеточием (:), но иногда удобно задать несколько операторов по одному на строке, но так, чтобы при нажатии клавиши <Enter> они одновременно были переданы символьному анализатору на выполнение. Это осуществляется нажатием комбинации клавиш <Shift>+<Enter> по за-

вершении ввода оператора. В этом случае введенный оператор не пересылается на обработку интерпретатору, а система ожидает продолжения ввода пользователя, автоматически переместив курсор на новую строку, причем в новой строке приглашение на ввод не появляется. Таким способом можно ввести несколько операторов по одному на каждой строке и после нажатия клавиши <Enter> все введенные операторы последовательно будут выполнены. На рис. 1.6 демонстрируется рабочий лист, на котором использована подобная техника ввода команд.

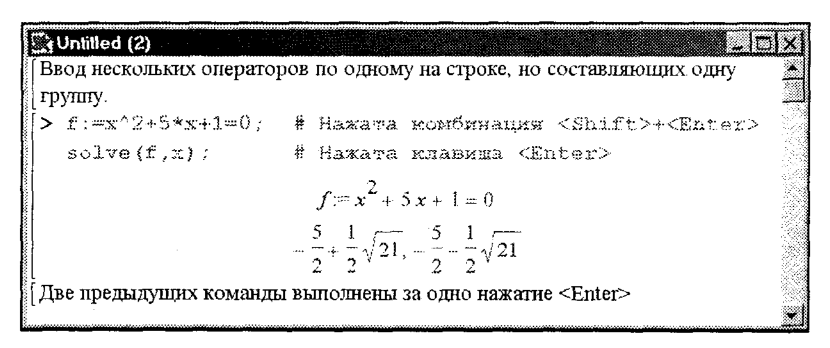


Рис. 1.6. Несколько команд в одной группе

Введенная таким способом последовательность команд образует одну группу, структурную единицу рабочего листа, с которой можно осуществить определенные действия: выполнить за одно нажатие клавиши <Enter>, объединить с другими группами или, наоборот, разбить большую группу на более мелкие. Отметим, что в группу кроме самих команд входит и их вывод, что отмечается приложением Maple слева одной большой квадратной скобкой.

Относительно области ввода следует еще сказать следующее. Когда сохраненный рабочий лист открывается в приложении Maple, то курсор устанавливается на первую область ввода (группу), в которой находятся команды. При нажатии клавиши <Enter> команды этой группы выполняются, а курсор перемещается на следующую группу с командами, пропуская все расположенные между ними группы с текстовыми комментариями.

1.2.2. Область вывода

Результаты выполнения операторов, введенных в области ввода, отображаются в области вывода. Для неграфического вывода Maple позволяет задать три формата: в виде *строковой записи*, аналогичной формату ввода команд в нотации Maple, в виде *символьной записи*, имитирующей запись формул в математической нотации, и в виде обычной *математической нотации*, использующейся при наборе математических формул в издательском деле. При выводе результата выполнения команды в принятой математической нотации возможны два варианта. В первом случае отображенный вывод

нельзя редактировать, а во втором — можно. Это означает, что при выделении области вывода в первом случае выделяется все выражение целиком, а в случае редактируемого вывода можно выделить отдельные элементы выражения и произвести над ними определенные действия, например, скопировать путем перетаскивания мышью в строку ввода. Все четыре типа формата вывода показаны на рис. 1.7, причем для редактируемого вывода в форме математической нотации выделен отдельный элемент полученного выражения.

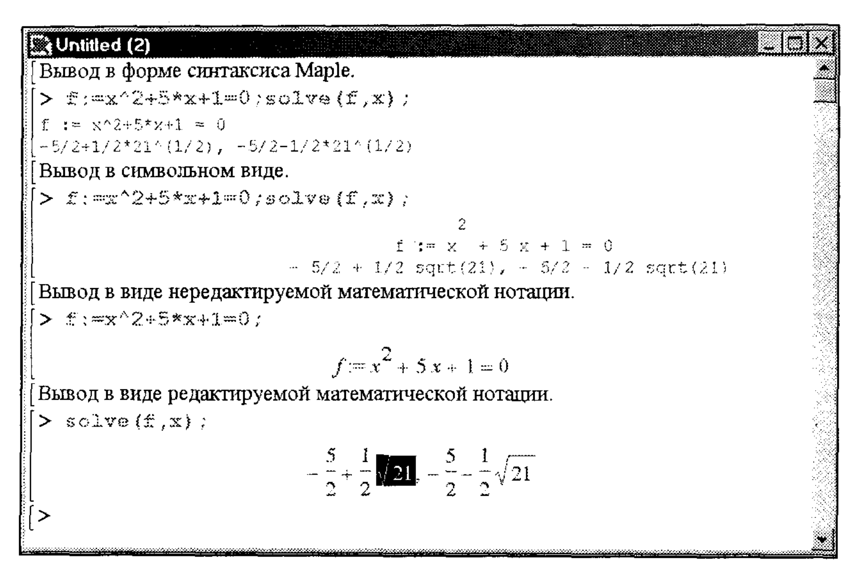


Рис. 1.7. Различные форматы вывода результатов команды

Установить один из возможных вариантов отображения информации в области вывода можно командами подменю **Output Display** меню **Option**. Для строкового вывода следует воспользоваться командой **Maple Notation**, вывод в форме символьной записи устанавливается командой **Character Notation**, не редактируемый вывод в виде математической нотации — командой **Typeset Notation** и, наконец, редактируемая математическая нотация — командой **Standard Math Notation**, которая установлена в Maple по умолчанию.

1.2.3. Вывод графики

Вывод графики в Maple можно осуществлять непосредственно в рабочий лист (режим по умолчанию) или в отдельное окно. Задание соответствующих режимов осуществляется командами **Inline** и **Window** подменю **Plot Display** меню **Options**. Оба типа вывода графики представлены на рис. 1.8.

На рисунке показан рабочий лист, в котором два оператора выводят графики одной и той же функции $y = \sin(x) + \cos(3\,x)$. Отличие заключается в том, что первый оператор `plot()` выполняется при включенном режиме вы-

вода графики в рабочий лист, а второй оператор `plot()` работает, когда графика выводится в окне, в данном случае в окне **Untitled (3)**. Обратим внимание на то, что содержимое строки основного меню и контекстной панели инструментов отличаются от соответствующего меню и панели инструментов на предыдущих рисунках. Это связано с тем, что окно графики в данный момент является активным — заголовок окна имеет яркий цвет. Поэтому эти элементы интерфейса пользователя отображают команды, связанные с обработкой графики в окне.

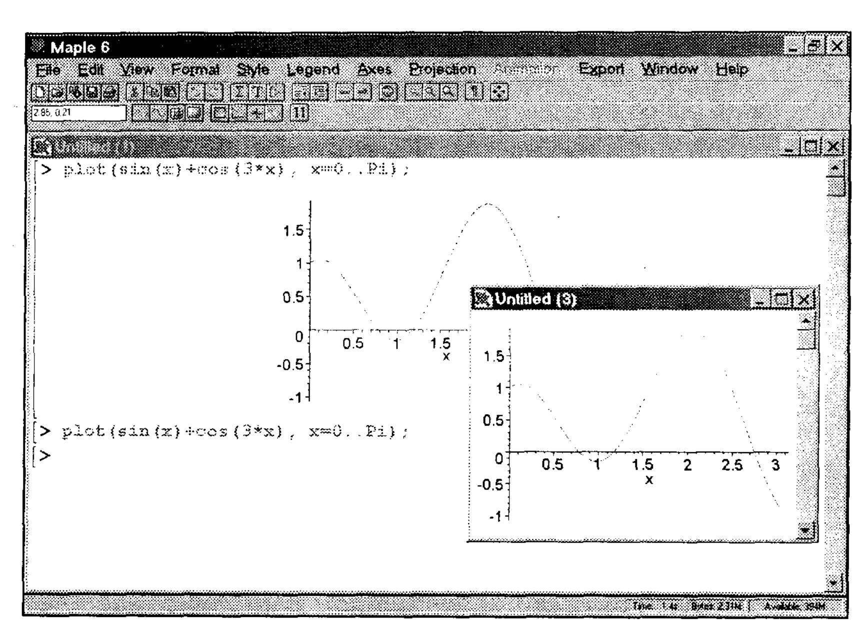


Рис. 1.8. Два режима вывода графики

Вывод графики в отдельном окне полезен для отображения промежуточных результатов расчетов, так как графические изображения, расположенные на рабочем листе, значительно увеличивают размер файла при сохранении и требуют дополнительной оперативной памяти при открытии документа с графикой.

1.3. Палитры, электронные таблицы, контекстные меню

1.3.1. Работа с палитрами

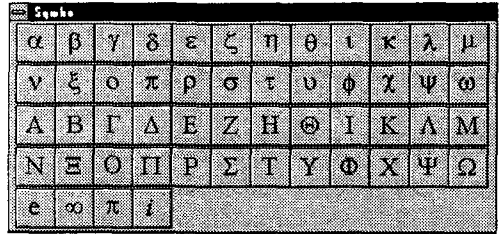
Пользователь вводит команды Maple в строке ввода, используя клавиатуру. При этом ему приходится запоминать, особенно при первом ознакомлении с системой, достаточно большое их количество. В интерфейс пользователя

в версии Maple V Rel5 были введены *палитры* — небольшие окошки с набором шаблонов для ввода определенных команд и объектов Maple, которые можно было отобразить и постоянно использовать на протяжении всего сеанса работы в Maple, закрывая их при необходимости. В версии Maple 6 такое средство ввода наиболее часто используемых команд немного улучшено. Пользователи предыдущей версии Maple, наверное, согласятся с тем, что использование палитр в ней было достаточно неудобно из-за их небольшого размера и слишком мелких надписей на их кнопках. В последней версии это неудобство исправлено. Теперь пользователь может отображать палитры в двух видах: с таким же размером, как и в предыдущей версии, и в увеличенном в 1,5 раза.

Всего Maple предлагает три вида палитр:

- Палитру для ввода символов (команда **View ➢ Palettes ➢ Symbol Palette**).
- Палитру с шаблонами для ввода выражений Maple (команда **View ➢ Palettes ➢ Expression Palette**).
- Палитру с шаблонами для ввода матриц размерности не более (4×4) (команда **View ➢ Palettes ➢ Matrix Palette**).

В Maple 6 в подменю **View ➢ Palettes** добавлены еще две команды: отобразить все палитры **Show All Palettes** и скрыть все палитры **Hide All Palettes**. Внешний вид палитр показан на рис. 1.9.



Палитра для ввода символов


Палитра ввода матриц

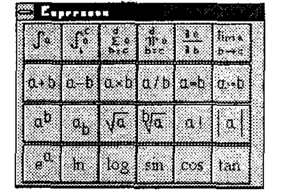

Палитра для ввода выражений

Рис. 1.9. Внешний вид палитр Maple

Размеры панелей палитр устанавливаются командами подменю **Options ➢ Palette Size**, при выборе которых они отображаются малого размера (**Small**), большого размера (**Large**), или размеры выбираются приложением Maple 6 в зависимости от размеров экрана монитора компьютера (**Best Choice**).

Работа с палитрами достаточно проста. Если пользователь желает использовать в своей работе греческие буквы и ему не хочется, или он забыл, как набивать их названия на английском языке, то символьная палитра поможет ему в этой работе. На ней расположен ряд кнопок, нажатие которых вставляет в рабочий лист в точку ввода (место, где расположен курсор) правильное написание греческой буквы, соответствующей синтаксису языка Maple.

Например, нажатие кнопки приведет к вставке наименования греческой строчной буквы альфа с завершающей точкой с запятой, если курсор находился в области ввода:

```
> alpha;
```

Нажав <Enter> и тем самым отправив команду на выполнение интерпретатору Maple, мы увидим отображение греческой буквы альфа в области вывода:

$$\alpha$$

Если при нажатии кнопки с буквой в области ввода часть выражения была выделена, то она заменится на соответствующую букву.

Если курсор находился в области вывода, в котором выделенным была часть математического выражения, то при нажатии на кнопку с буквой альфа непосредственно после области вывода будет сформирована строка ввода с командой, соответствующей математическому выражению, в котором выделенная часть заменена греческим символом альфа. Например, если область вывода содержит присваивание некоторой переменной `f` функции `sin(x)` с выделенным наименованием функции

$$f{:=}\sin(x)$$

то выполнение указанных выше действий приведет к появлению следующей строки ввода на рабочем листе:

```
> f := alpha(x);
```

Замечание

Аналогичным образом работают и другие палитры при выделении частей математических выражений в области вывода. Небольшое отличие заключается в том, что две другие палитры не заменяют выделенное выражение, а используют его в качестве значения первого элемента матрицы в случае палитры ввода матриц, или в качестве первого аргумента соответствующей функции палитры ввода выражений.

Кроме букв греческого алфавита последнюю строку данной палитры можно использовать для ввода нескольких математических величин: чисел e и π, бесконечности ∞, а также мнимой единицы i.

Палитра ввода матриц позволяет быстро ввести шаблон для создания одномерных и двумерных матриц, протяженности каждой размерности которых не превышают числа четыре. При нажатии, например, на кнопке , представляющей шаблон квадратной матрицы размерности 2×2, в строку ввода будет вставлена команда создания матрицы:

```
> Matrix([[%?, %?], [%?, %?]]);
```

в которой на местах расположения элементов матрицы находятся специальные символы, так называемые *шаблоны-заполнители*. Для задания конкрет-

ных значений элементов их следует заменить на те величины, которые требуются, например, по условию решения задачи. Объект, создаваемый при выполнении указанной команды, представляет матрицу из нового пакета линейной алгебры LinearAlgebra.

Замечание

В версии Maple V Rel5 палитра ввода матриц реализовала команду matrix() из пакета линейной алгебры linalg, который присутствует и в новой версии Maple 6, однако разработчики системы аналитических вычислений рекомендуют пользоваться новым пакетом линейной алгебры LinearAlgebra, который реализован на основе новой технологии внедрения в систему аналитических вычислений программ, написанных на компилируемых языках. (Возможности нового пакета LinearAlgebra и технологию работы с ним *см. в гл. 3.*)

Последняя палитра содержит шаблоны для ввода наиболее часто используемых математических операций при работе с символьными, и не только, выражениями. С ее помощью можно вставить в рабочий лист шаблоны для вычисления неопределенного и определенного интеграла от функции одной переменной, суммы и произведения индексных величин, производной и предела функции одной переменной. Здесь же находятся шаблоны для выполнения арифметических операций (сложение, вычитание, произведение, деление), вычисления степенных и показательных функций и многое другое.

При использовании шаблона для ввода необходимого выражения после нажатия на соответствующую кнопку палитры в точке ввода отображается команда Maple, содержащая вместо некоторых параметров шаблоны-заполнители, в которые следует подставить реальные значения, требуемые по ходу решения задачи. Например, шаблон для ввода суммы индексных величин выглядит так:

```
> sum(%?, %?=%?..%?);
```

Здесь вместо первого заполнителя (%?) следует подставить (выделив шаблон и набрав формулу на клавиатуре) необходимое индексное выражение, вместо второго заполнителя задать идентификатор индекса, а далее указать диапазон его изменения.

Все сказанное относится к случаю, когда установлен режим отображения команд в форме синтаксиса Maple. Если пользователь использует для этих целей режим отображения математической нотации, то процедура заполнения шаблона ничем не отличается от только что описанной. Разница заключается лишь во внешнем виде встраиваемого в страницу шаблона. Например, шаблон суммы в этом случае будет выглядеть так:

$$> \sum_{? = ?}^{?} ?$$

1.3.2. Электронная таблица

Очень часто при решении математических задач возникает необходимость создать таблицу значений некоторой функции. Для решения этой проблемы Maple предлагает использовать объект *электронная таблица*, которая встраивается в рабочий лист по команде **Insert ➢ Spreadsheet**. При этом становится доступным меню **Spreadsheet** основного меню Maple, и на контекстной панели инструментов отображаются кнопки команд, предназначенных для работы с электронной таблицей. Внешний вид электронной таблицы показан на рис. 1.10 и не отличается от вида электронных таблиц, предоставляемых другими приложениями, например, MS Excel, хотя их функциональность и технология работы с ними несколько различаются, хотя бы в том, что в ячейках электронной таблицы Maple можно задавать и вычислять все его команды.

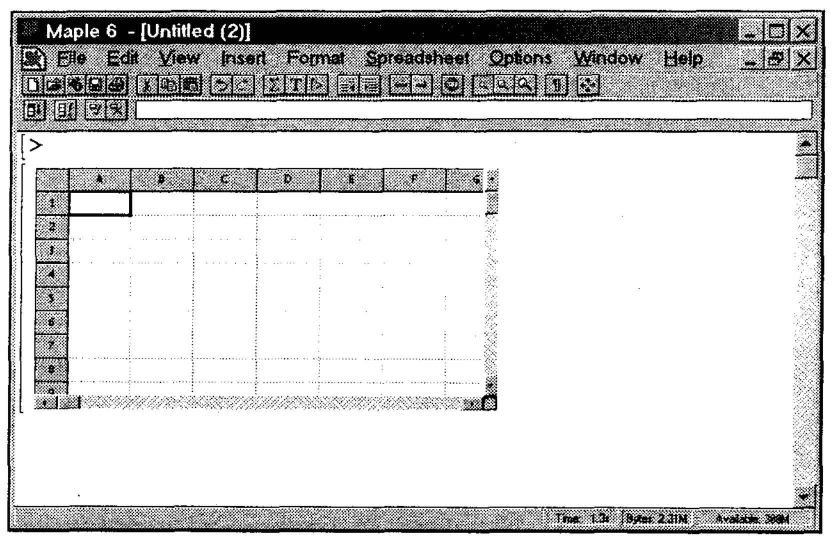


Рис. 1.10. Внешний вид электронной таблицы Maple

Если в электронной таблице выделена ячейка, то ввод с клавиатуры направляется именно в нее, хотя непосредственно в ее поле он не отображается. Все, что мы вводим в этом случае, отображается в поле ввода контекстной панели инструментов, а сама ячейка заштриховывается перекрещивающимися линиями, как показано на рис. 1.11. Для отображения введенной с клавиатуры информации в самой ячейке по окончании ввода следует нажать клавишу <Enter> или одну из клавиш со стрелками. В случае нажатия <Enter> активной становится ячейка снизу, а при завершении ввода клавишами со стрелками активной становится соседняя ячейка в соответствующем направлении.

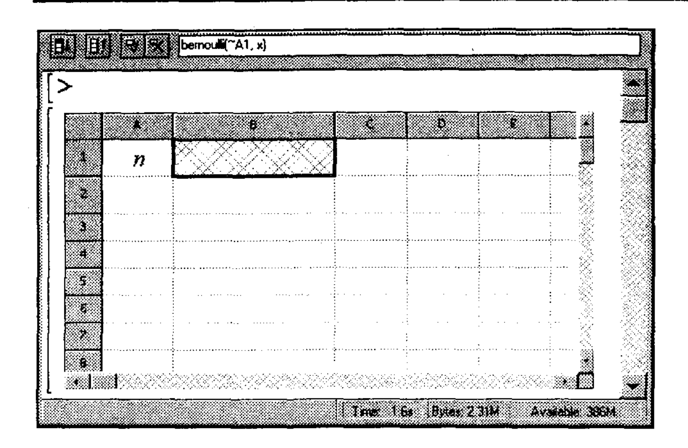


Рис. 1.11. Ячейки с введенным значением (A1) и в процессе ввода значения (B1)

Замечание

Для ввода содержимого Буфера обмена в ячейку электронной таблицы следует явным образом установить курсор в поле ввода контекстного меню. В противном случае информация будет вставлена в рабочий лист перед электронной таблицей.

Как и в других системах электронных таблиц, каждая ячейка имеет собственный адрес, который складывается из наименования столбца (прописные латинские буквы) и номера строки, в которых она расположена. Например, ячейка на рис. 1.11, содержащая *n*, имеет адрес A1, ячейка, в которую осуществляется ввод, — A2 и т. д. Чтобы сослаться на содержимое ячейки, следует перед ее адресом поставить символ тильда (~).

Предположим, что перед нами поставлена задача вычислить все полиномы Бернулли до 5 степени включительно. Для этого в ячейку A1 введем *n*, а в ячейку B1 — команду вычисления полинома Бернулли, в которой первый параметр, определяющий степень полинома, зададим в виде ссылки на содержимое ячейки A1:

```
bernoulli(~A1,x)
```

В ячейку A2 введем число 1 и выделим ячейки с A2 по A6. (Последнее действие осуществляется перемещением вниз указателя мыши при нажатой правой кнопке.) Для их заполнения нажмем кнопку ▦ контекстной панели инструментов. Отобразится диалоговое окно заполнения диапазона ячеек, показанное на рис. 1.12, в котором в группе **Direction** выберем переключатель **Down**, в поле **Step Size** зададим 1 в качестве шага изменения значений при заполнении ячеек выделенного диапазона, и нажмем кнопку **OK**.

В результате выполнения перечисленных действий ячейки с A3 по A6 заполнятся числами 2, 3, 4 и 5.

Выделим теперь диапазон ячеек с B1 по B6. Снова отобразим диалоговое окно заполнения диапазона **Fill**, но теперь установим в нем только пере-

ключатель **Down**. Нажмем кнопку **OK** и увидим, как ячейки выделенного диапазона будут заполняться явным видом полиномов Бернулли степени, соответствующей содержимому ячеек столбца A. Результат выполнения всех наших действий показан на рис. 1.13.

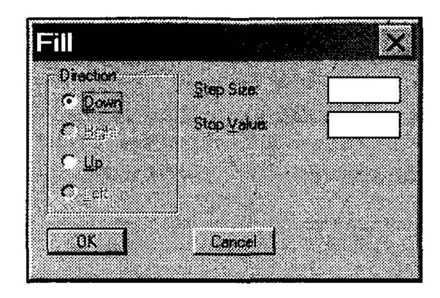


Рис. 1.12. Диалоговое окно заполнения диапазона ячеек

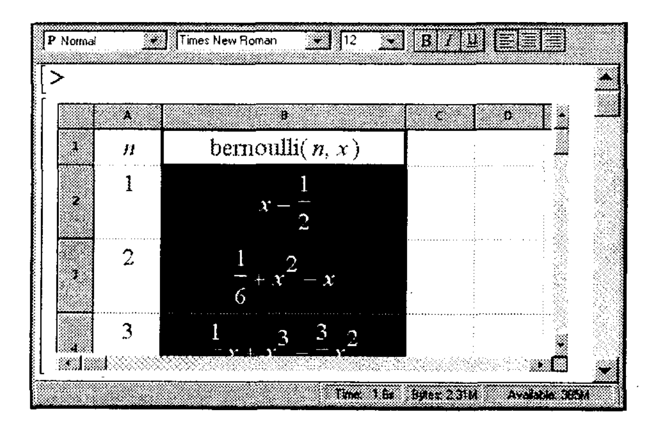


Рис. 1.13. Вычисление полиномов Бернулли в электронной таблице

При заполнении ячеек столбца B полиномами Бернулли последние вышли из области отображения электронной таблицы, которая была вставлена с размерами по умолчанию, и перестали быть видимыми. Чтобы увидеть их полностью, следует увеличить размеры электронной таблицы. Для этого сначала надо увеличить размеры самого рабочего листа, чтобы он заполнял все рабочее пространство окна Maple, щелкнув на кнопке **Развернуть** (стандартная кнопка интерфейсов программ, работающих под управлением операционной системы Windows) в правой части заголовка окна рабочего листа, если окно рабочего листа еще не развернуто. Для увеличения размеров электронной таблицы ее необходимо выделить, подведя указатель мыши к границе электронной таблицы, и щелкнуть левой кнопкой. После этого следует прихватить мышью один из небольших черных прямоугольников, расположенных по периметру выделения и называемых маркерами размера, и передвинуть его для увеличения размеров электронной таблицы (рис. 1.14).

Есть еще один способ увеличить размеры электронной таблицы. Если подвести указатель мыши к нижнему правому углу электронной таблицы, то он

изменится на двустороннюю стрелку. После этого можно, удерживая нажатой левую кнопку, перемещать указатель мыши в нужном направлении, чтобы увеличить или уменьшить размеры электронной таблицы.

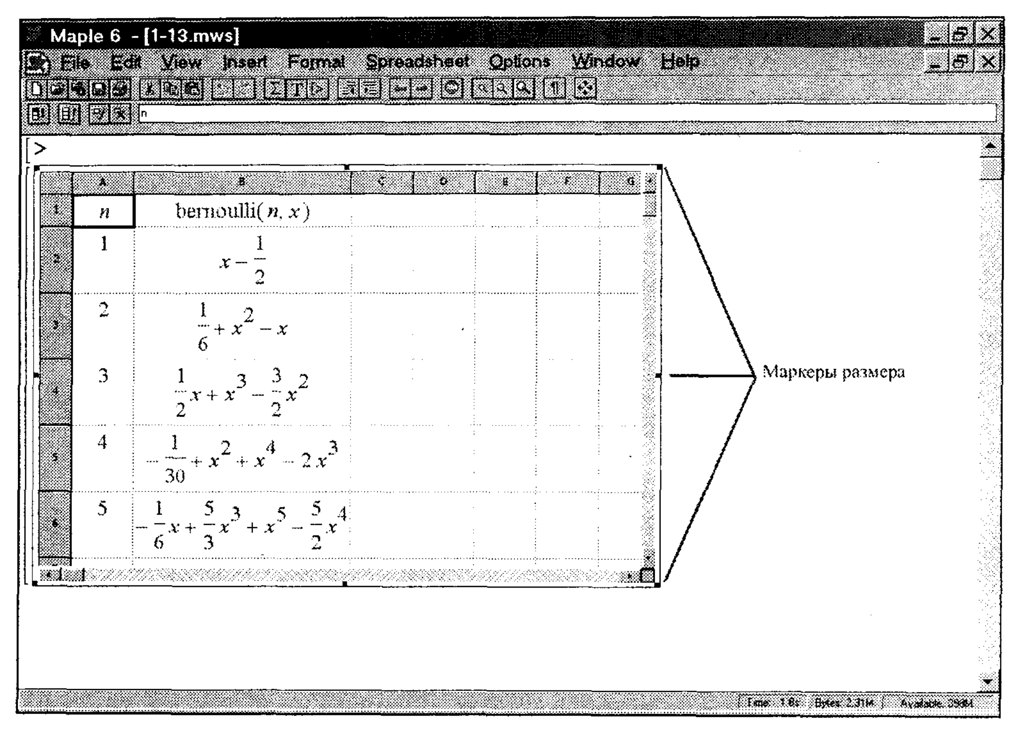


Рис. 1.14. Выделенная электронная таблица с маркерами размера

Замечание

Заполнить ячейки электронной таблицы можно не только с помощью кнопки контекстной панели инструментов электронной таблицы, но и с помощью команды **Spreadsheet** ➢ **Fill** ➢ **Detailed**, которая доступна при выделенном диапазоне ячеек и отображает диалоговое окно заполнения ячеек **Fill**.

Используя механизм ссылки на ячейки и их заполнения, можно вычислить, например, интегралы от полиномов Бернулли. Для этого следует в ячейке C1 ввести команду вычисления интеграла, в которой первый параметр является ссылкой на содержимое ячейки B1:

```
> int(~B1,x)
```

После этого выделить диапазон ячеек B1:B6 и выполнить команду **Spreadsheet** ➢ **Fill** ➢ **Down** или нажать кнопку заполнения ячеек на контекстной панели инструментов. В соответствующих ячейках будут вычислены инте-

гралы от полиномов Бернулли, соответственно, первой, второй и т. д. степеней до пятой включительно.

На примере вычисления полиномов Бернулли и их интегралов показано использование электронных таблиц Maple для быстрого создания таблиц, но этим не исчерпываются возможности работы с электронными таблицами. Если изменить значение влияющей ячейки, т. е. ячейки, на которую ссылаются другие ячейки и которые называют зависимыми, то последние будут заштрихованы, предупреждая пользователя, что их текущие значения не соответствуют текущему значению влияющей ячейки. Подобная ситуация отражена на рис. 1.15 при изменении значения ячейки A6 с 5 на 7.

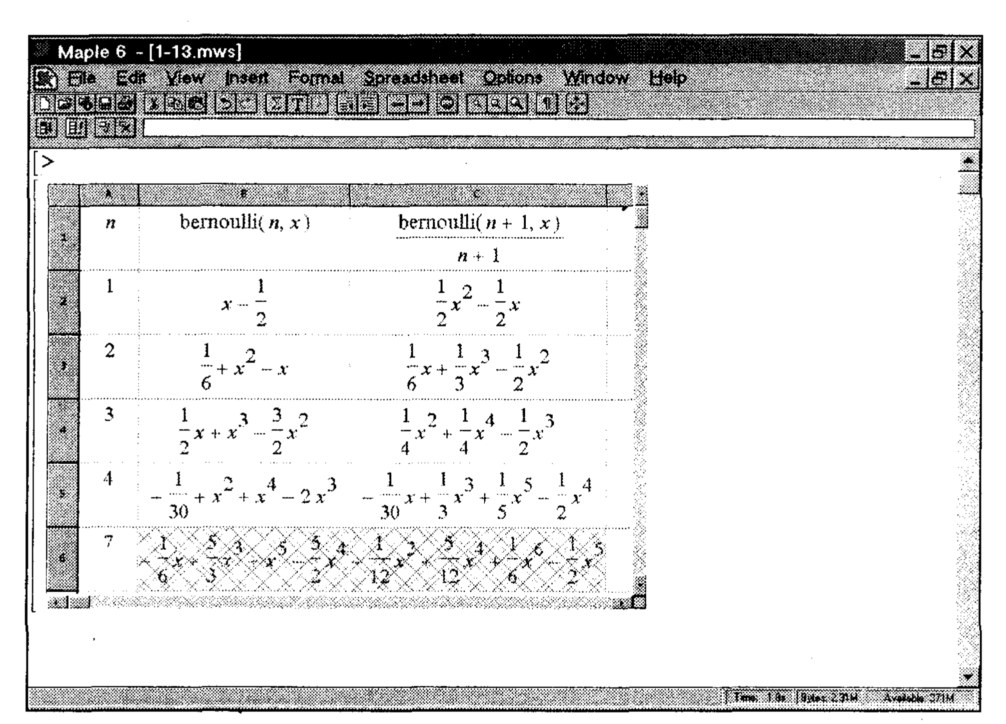


Рис. 1.15. Индикация несоответствия значений влияющих и зависимых ячеек

Чтобы привести в соответствие значение влияющей ячейки и содержимого зависимых от нее ячеек, следует либо нажать кнопку вычисления ячеек с устаревшими значениями, либо выделить в электронной таблице диапазон заштрихованных ячеек и выполнить команду **Spreadsheet ➤ Evaluate Selection**, либо, не делая никаких выделений, выполнить команду **Spreadsheet ➤ Evaluate Spreadsheet**, которая, правда, заново вычислит значения всех зависимых ячеек в электронной таблице.

Maple предоставляет возможность изменять внешний вид ячеек электронных таблиц. Команды **Row ➤ Height** и **Row ➤ Width** меню **Spreadsheet** служат,

соответственно, для явного задания высоты и ширины ячеек, а команда **Properties** этого же меню отображает диалоговое окно **Spreadsheet Properties** (рис. 1.16), которое позволяет установить широкий спектр свойств электронной таблицы.

Переключателями группы **Alignment** можно установить выравнивание содержимого ячейки, группа **Precision** позволяет задать количество значащих цифр в дробной части чисел с плавающей точкой при выполнении вычислений и при отображении их в ячейке таблицы. В раскрывающемся списке **Cell Color** можно выбрать предлагаемый приложением Maple или задать собственный цвет фона ячеек, а переключатели группы **Evaluation** позволяют установить режим вычисления электронной таблицы: либо символьный (режим по умолчанию), либо численный с плавающей точкой. И, наконец, поле **Spreadsheet Name** задает имя электронной таблицы, по которому к ней можно ссылаться и выполнять определенные действия из программы, написанной на языке Maple.

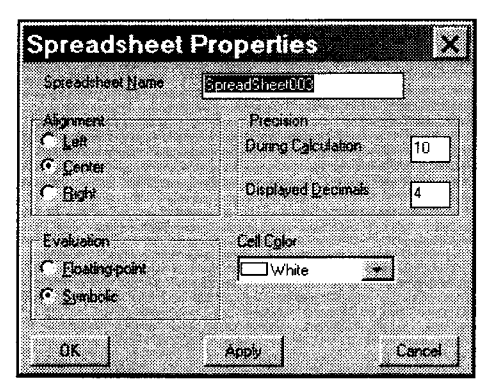


Рис. 1.16. Диалоговое окно **Spreadsheet Properties** установки свойств электронной таблицы

Замечание

Возможность программного управления электронной таблицей и адресация ее ячеек из программы Maple является новой функциональной возможностью, добавленной в Maple 6 и отсутствовавшей в предыдущих версиях. Для использования этой возможности в нее включен новый пакет `Spread`, предоставляющий программный интерфейс для взаимодействия с электронными таблицами.

В новую версию Maple 6 добавлена возможность экспортировать содержимое электронной таблицы в формат программ Matlab и MatrixMarket, а также в формат, разделителем между данными в котором служит символ табуляции. Этим целям служат, соответственно, три команды **Matlab**, **MatrixMarket** и **Tab Delimited** подменю **Export** и **Import** меню **Spreadsheet**. Если в электронной таблице выделен диапазон ячеек, то экспортируются только данные из этого диапазона, в противном случае все данные из всех заполненных ячеек. При импортировании, если выделен диапазон ячеек, то только в них импортируются данные из внешних файлов. Если в файле больше данных, чем ячеек в диапазоне, то оставшиеся данные отсекаются и не импортируются. В противном случае содержимое файлов импортируется полностью.

1.3.3. Обмен данными и контекстные меню

Для обмена данными между приложением Maple и другими функционирующими в среде Windows программами можно использовать Буфер обмена операционной системы. Команды **Cut** и **Copy** меню **Edit** вырезают и копируют выделенный фрагмент рабочего листа или весь рабочий лист в Буфер обмена. По умолчанию информация сохраняется в формате RTF, что позволяет вставить ввод/вывод Maple в стандартной математической нотации в виде рисунков в документ MS Word, сохраняя при этом полностью внешний вид рабочего листа, в том числе и выбранные цвета для шрифтов разных стилей Maple. Команда **Paste** этого же меню предназначена для вставки содержимого Буфера обмена в рабочий лист. Для вызова всех указанных команд можно использовать соответствующие комбинации клавиш, которые отображаются рядом с командой в раскрывающихся меню. В системе Windows можно также использовать <Ctrl>+<Insert> для копирования в Буфер обмена, <Shift>+<Insert> — для вставки и <Shift>+<Del> — для вырезания.

Командой **Copy as Maple Text** можно скопировать рабочий лист или его фрагмент в формате текста Maple, специальном текстовом формате, сохраняющем структуру рабочего листа. В этом формате текстовая область ввода рабочего листа предваряется символом #, перед содержимым области ввода команд ставится символ >, а область вывода представлена в виде символьной записи. Если в Буфере обмена содержится информация в формате текста Maple, то команда **Paste Maple Text** вставляет его, преобразуя в текстовые области и области ввода команд рабочего листа.

Замечание

Maple автоматически распознает содержимое Буфера обмена в формате текста Maple, делая доступным команду вставки в этом формате. Эту же информацию можно вставить и командой **Paste**, однако, в этом случае ее преобразование в структуру рабочего листа не произойдет.

Замечание

Технологию копирования и вставки из Буфера обмена можно использовать не только для обмена информацией с другими программами, но и для обмена информацией между рабочими листами сеанса Maple, а также копирования и вставки команд в разные части одного рабочего листа. Электронные таблицы Maple также поддерживают технологию работу с Буфером обмена.

Копировать информацию на одном рабочем листе можно с помощью технологии "перетаскивания" мышью, так называемой технологией "drag-and-drop". Любой выделенный фрагмент рабочего листа (область ввода или ее часть, область вывода или ее часть) можно скопировать и переместить в любое место рабочего листа, наведя на него указатель мыши (при этом он изменится и примет вид курсора) и при нажатой левой кнопке перемещать в нужное место. При этом курсор примет обычный вид с небольшим прямо-

угольником в своей нижней части. Подведя его к требуемому месту вставки, следует отпустить кнопку мыши. Выделенный фрагмент переместится в указанное новое место. Если при перетаскивании нажать клавишу <Ctrl>, то выделенный фрагмент скопируется в соответствующее место, а при перетаскивании рядом с курсором появится знак плюс (+).

Еще одно удобное средство работы в интерактивном режиме с объектами Maple рабочего листа, — *контекстные меню*, панели с набором команд, которые отображаются нажатием правой кнопки мыши при наведении указателя мыши на какой-либо объект области ввода или вывода рабочего листа. Для области ввода, содержащей текст или команды, набор команд в соответствующих контекстных меню предопределен, и его нельзя изменить. Для объектов области вывода содержимое контекстных меню варьируется для разных объектов и с помощью пакета context его можно изменять или создавать новые контекстные меню для созданных пользователем объектов, отображаемых в области вывода. Контекстные меню для текста и команд Maple, расположенных в области ввода или встроенных в текстовую область рабочего листа, показаны на рис. 1.17.

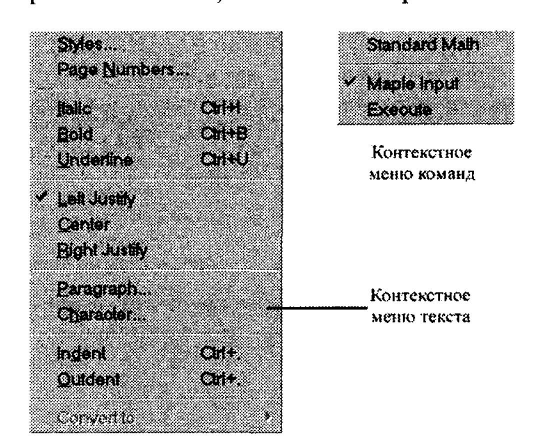


Рис. 1.17. Контекстные меню для текста и команд Maple

Контекстные меню очень удобны при интерактивной работе. Например, если надо быстро начертить график функции, то следует в области ввода просто набрать команду, задающую необходимую функцию, выполнить ее и отобразить контекстное меню области вывода, как показано на рис. 1.18.

Обратите внимание на разнообразие возможных команд для данной области вывода: интегрирование (**Integrate**), дифференцирование (**Differentiate**), разложение на множители (**Factor**), построение на его основе новых конструкций Maple (**Constructions**) и т. д. Если для команды следует уточнить некоторый параметр, например указать переменную интегрирования для команды вычисления интеграла, или команда содержит набор возможных команд, т. е. является подменю, как команда **Constructions**, то справа от нее расположен треугольник, означающий, что при ее выборе раскроется подменю

параметров или команд. Если в контекстном меню, представленном на рис. 1.18, выбрать команду **Plots ➢ 2-D Plot**, то с помощью специальной команды `smartplot()` будет построен график функции, содержащейся в области вывода (рис. 1.19).

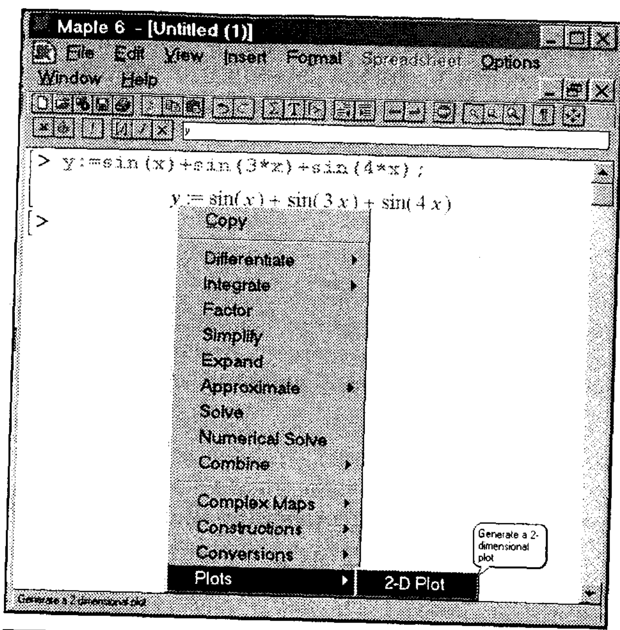


Рис. 1.18. Контекстное меню для области вывода, содержащей результат команды присваивания

Рис. 1.19. График функции, построенный с помощью команды контекстного меню

Команда `smartplot()` использует при построении графика функции значения параметров по умолчанию, которые опять-таки можно изменить с помощью контекстного меню полученного графика.

И здесь следует сказать об очень полезном свойстве графика, построенного функцией `smartplot()`: он позволяет добавлять графики других функций простым перетаскиванием их определения из области вывода в область графика. Например, если необходимо добавить график функции $y = \sin(x)$, то достаточно выполнить команду

```
> sin(x);
```

выделить полученную область вывода и перетащить ее мышью в область графика. Результат показан на рис. 1.20.

Замечание

Если необходимо добавить график функции и оставить определение функции в области вывода, то перетаскивание следует осуществлять при нажатой клавише <Ctrl>. В противном случае из области вывода перетаскиваемая выделенная часть будет удалена.

Аналогичным перетаскиванием можно удалить какую-либо функцию из области графика. Для этого следует подвести указатель мыши на линию графика функции и переместить его за пределы рисунка при нажатой левой кнопке мыши. При этом в том месте рабочего листа, куда он будет перемещен, появится команда, задающая удаленную функцию.

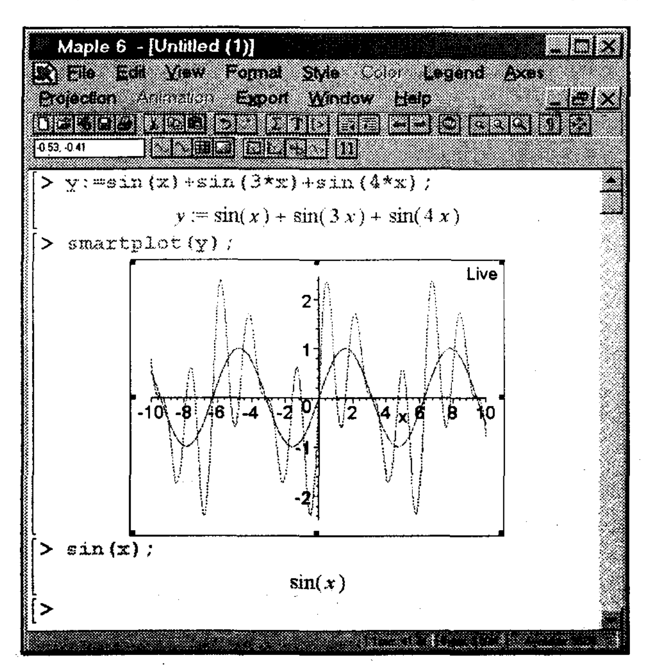


Рис. 1.20. Добавление новой функции на график, построенный командой `smartplot()`

Замечание

Если при перетаскивании из области графика линии функции удерживать клавишу <Ctrl>, то график этой функции не будет удален, но в месте, куда он был перетащен, появится команда, определяющая вид функции.

1.4. Работа с меню

Интерфейс системы Maple является многодокументным интерфейсом. Это означает, что в одном сеансе можно открыть несколько документов Maple и с помощью команд меню **Window** переключаться между ними или отобразить все окна открытых документов.

Концепция интерфейсов, поддерживающих работу со многими документами, так называемых *MDI-интерфейсов* (от английского Multiple Document Interface), предполагает, что при работе с разнородными документами основное меню может изменяться, чтобы предоставить пользователю команды, специфичные для того типа документа, с которым в данный момент он работает. В Maple существует три типа документов: рабочий лист, графический вывод (если задан режим вывода графики в отдельном окне) и лист Справки. С каждым из них связаны свои списки меню, которые отображаются в строке меню при активизации соответствующего документа.

Как уже говорилось, при выделении графических объектов основное меню меняется на меню работы с графикой.

Итак, в строке меню одновременно может отображаться только один список меню, набор элементов которого меняется и зависит от того, в какой области рабочего листа находится в данный момент курсор, а также является ли активной в данный момент страница Справки Maple или окно графического вывода. Всего может быть отображено пять видов основного меню:

- Стандартное меню рабочего листа
- Меню электронной таблицы
- Меню двумерной графики
- Меню трехмерной графики
- Меню справочной системы

Отметим, что хотя в списке меню для двумерной и трехмерной графики некоторые названия меню совпадают, сами меню отличаются, так как содержат различные списки команд.

В этом разделе мы остановимся только на стандартном меню рабочего листа. Меню электронной таблицы **Spreadsheet** мы практически полностью разобрали в разделе 1.3.2, графические меню будут описаны в главе 4, посвященной работе с графикой в Maple, а справочная система и его меню рассматривается в последнем разделе данной главы.

1.4.1. Структура меню

При выборе какого-либо заголовка в основном меню щелчком мыши отображается меню с набором доступных команд и заголовков подменю. Команда отличается от подменю тем, что она сразу же выполняется при щелчке на ней кнопкой мыши, а выбор подменю сопровождается отображением дополнительного списка команд. На рис. 1.21 показана панель со списком команд меню **File**.

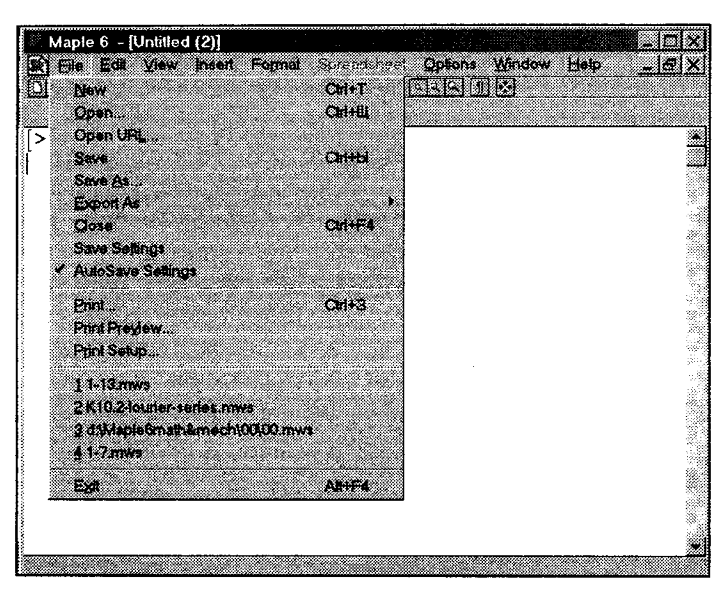


Рис. 1.21. Команды меню **File**

На примере меню **File** объясним структуру раскрывающегося списка команд и подменю.

В любом меню в левом столбце расположены имена (названия) команд, а справа соответствующие каждой команде "горячие клавиши" — комбинации клавиш, одновременное нажатие которых приводит к выполнению команды. Например, если на клавиатуре нажать одновременно клавиши <Ctrl> и <T> (такие действия принято обозначать <Ctrl>+<T>), то выполнится команда **New** меню **File**. Обратим внимание, что не все команды имеют быстрые клавиши, например команда **Save As**.

Если в правом столбце напротив имени команды отображается символ ▸, это означает, что данная строка является заголовком подменю, и при ее выборе отобразится дополнительная панель с командами (а может быть и с новыми подменю).

Если в названии команды присутствует многоточие, это означает, что для выполнения данной команды требуется дополнительная информация, кото-

рую пользователь должен ввести в отображаемом командой диалоговом окне. Например, для команды **Save As** (Сохранить как) необходимо в диалоговом окне задать имя файла и его расположение (путь).

Обычно список команд разделен на логические группы, содержащие команды, выполняющие функционально-связанные действия. Такие группы отделяются в панели команд разделительной чертой. Например, команды меню **File**, связанные с печатью, выделены в одну группу.

Действия некоторых команд заключаются в установлении какого-либо режима работы системы Maple. Обычно они работают как переключатели: включают или выключают соответствующий режим. Если режим, определенный командой, включен, то слева от ее имени отображается метка ✔. Например, в меню **File** на рис. 1.21 включен режим **AutoSave Settings** (Автоматическое сохранение установок).

Выбор меню и их команд можно производить и с помощью клавиатуры. Если посмотреть на названия меню, то можно заметить, что в них один символ подчеркнут. Этот символ можно использовать в качестве "быстрых клавиш" для выбора меню. При нажатии комбинации клавиш <Alt>+ +<подчеркнутый символ> отобразится список команд меню, в имени которого соответствующий символ подчеркнут. Например, для меню **File** таким символом является <F> (регистр клавиатуры не имеет значения).

Для выбора команд и подменю из раскрывающегося списка меню можно также применять "быстрые клавиши". Они отличаются от горячих клавиш, которые работают всегда, тем, что действуют *только* при раскрытом меню. Например, если меню **File** раскрыто, то нажатие клавиши <N> приведет к выполнению команды **New** (Новый). Впрочем, по меню можно передвигаться и с помощью клавиш со стрелками.

Если в меню название команды отображается серым цветом, то эта команда в данный момент не доступна. Обычно в Maple такое происходит с командами вставки содержимого Буфера обмена или копирования в Буфер обмена. Если он пуст, то, естественно, ничего нельзя вставить в документ, и поэтому команды вставки из Буфера обмена не доступны. В случае если в рабочем листе нет выделенного фрагмента, то наоборот, ничего нельзя скопировать в Буфер обмена, и соответствующие команды недоступны.

1.4.2. Стандартное меню рабочего листа

Стандартное меню рабочего листа отображается при работе с рабочими листами и состоит из следующих меню: **File** (Файл), **Edit** (Правка), **View** (Вид), **Insert** (Вставка), **Format** (Формат), **Options** (Опции), **Window** (Окно) и **Help** (Справка). (Меню **Spreadsheet** опущено намеренно, так как оно недоступно, если только на рабочий лист не встроена электронная таблица.) Здесь кратко опишем основные, наиболее часто используемые при работе с Maple команды этих меню.

Замечание

В Maple 6 по сравнению с предыдущими версиями практически все меню реорганизованы: некоторые команды теперь входят в другие меню, в некоторых меню добавлены новые команды, отражающие введенную новую функциональность. Для пользователей версии Maple V Rel5 следует заметить, что все команды этой версии присутствуют и в новой версии, разве лишь находятся на новых местах.

1.4.2.1. Команды меню *File*

Команды этого меню предназначены для создания, открытия, сохранения и печати рабочих листов.

Новый рабочий лист создается при помощи команды **New** (Новый). Отметим, что при запуске Maple автоматически создает новый рабочий лист. Для сохранения рабочего листа в файле существуют две команды: **Save** (Сохранить) и **Save As** (Сохранить как). Первая сохраняет рабочий лист в том же самом файле, из которого рабочий лист и был открыт, а вторая в файле с другим именем. Команда **Save As** отображает стандартное диалоговое окно системы Windows для задания имени файла и его расположения на жестком диске.

Замечание

Некоторые команды меню дублируются кнопками основной панели инструментов. Например, создать новый рабочий лист можно нажатием кнопки .

Открыть сохраненный ранее в файле рабочий лист можно командой **Open** (Открыть). При этом в отображаемом диалоговом окне следует задать имя и расположение файла. Команда **Close** (Закрыть) закрывает текущий активный рабочий лист, спрашивая пользователя о подтверждении сохранения файла перед закрытием, если в рабочий лист были внесены какие-то изменения.

Рабочие листы сохраняются в файлах с расширением mws. Существует возможность сохранить рабочий лист в файлах других форматов. Для этого необходимо в диалоге команды **Save As** указать соответствующий формат, либо воспользоваться командами подменю **Export As** (Экспортировать в формате). Maple 6, как и предыдущая версия Maple V Rel5, позволяет сохранить рабочий лист как обычный текстовый файл (**Plain Text**) в текстовом формате Maple (**Maple Text**), формате издательской системы LaTeX (**Latex**) достаточно широко использующейся для написания математических текстов, и формате HTML (**HTML**).

В версию Maple 6 включены еще два формата: формат для обозревателя Maple (**Maple Explorer**) — специальной программы, позволяющей просматривать готовые файлы Maple, и формат RTF (**RTF**). Наличие преобразования в последний формат позволяет вставлять рабочие листы Maple в документы достаточно широко используемого текстового процессора MS Word.

В этом же меню существуют команды печати **Print** (Печать) и предварительного просмотра **Print Preview** (Предварительный просмотр печати) рабочего листа, а также команда **Printer Setup** (Установки принтера) для установки параметров принтера.

В связи с быстрым развитием и широким внедрением технологий Интернета в Maple 6 добавлена команда **Open URL**, которая позволяет открыть ресурс Интернета или файл Maple (с расширением mws), находящийся на другом компьютере в сети, не выходя из программы аналитических вычислений. Причем, если файл Maple открывается непосредственно в самом приложении, то произвольный ресурс Интернета открывается в окне программы Обозреватель Интернета, например, MS Internet Explorer или Netscape Navigator. В меню **Options** есть команда **Browser**, в диалоговом окне которой следует указать используемую для этих целей программу. На рис. 1.22 показан открытый в Maple 6 файл, расположенный по адресу:

www.maplesoft.com/apps/categories/education/appliedmath/worksheets /K10.2-fourier-series.mws

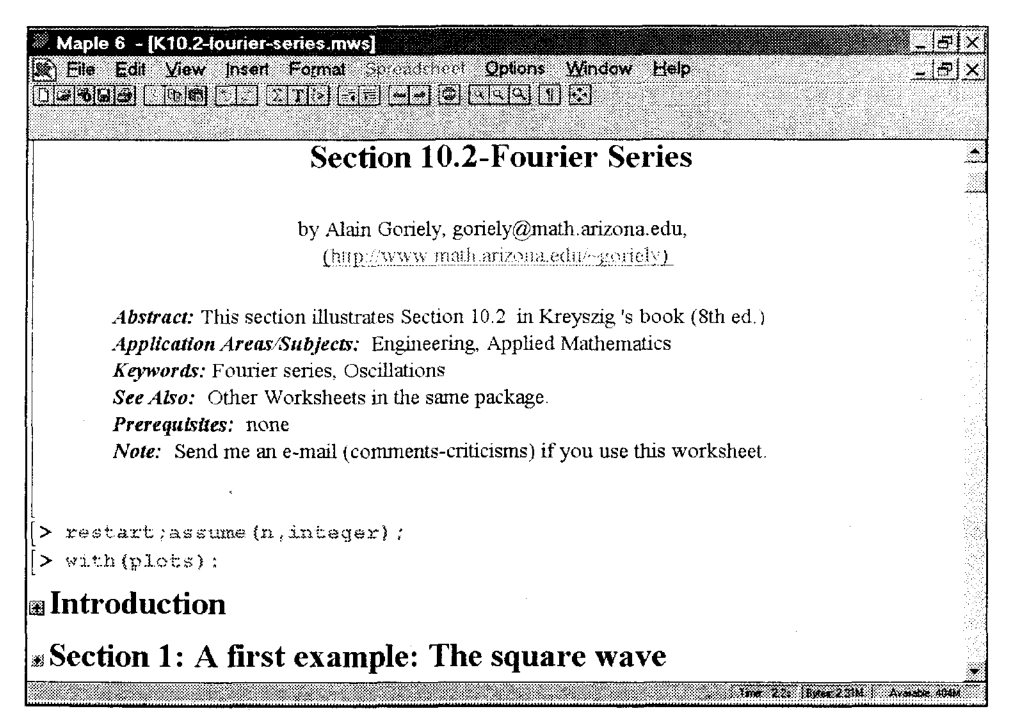


Рис. 1.22. Открытый в окне Maple 6 ресурс, расположенный в Интернете

Этот файл представляет собой один из разделов учебного пособия по численным методам для студентов, изучающих прикладную математику.

При открытии файла в Интернете в строке состояния отображается сначала сообщение о поиске соответствующего ресурса, а затем, если он найден, отображается в процентах процесс его загрузки на компьютер пользователя, например:

```
Retrieving worksheet: 120000 bytes of 240000 (50%)...
Загрузка рабочего листа: 120000 байт из 240000 (50%)...
```

После того как файл будет открыт командой **Save**, его можно сохранить для просмотра в автономном режиме.

Замечание

Подключение к Интернету компьютера пользователя следует осуществить до начала выполнения команды **Open URL**. Если компьютер не находится в сети, то эта команда просто выдает сообщение о не доступности заданного ресурса и не подсказывает осуществить соединение с Интернетом.

1.4.2.2. Команды меню *Edit*

Команды этого меню позволяют редактировать и вносить правку в рабочий лист. В основном они предназначены для работы с текстовыми комментариями, чтобы придать им привлекательный и удобочитаемый вид.

Большинство команд этого меню аналогичны командам меню **Edit** (Правка) других приложений Windows. Команды **Cut** (Вырезать), **Copy** (Копировать) и **Paste** (Вставить) позволяют соответственно вырезать или скопировать выделенный текст в Буфер обмена, а также вставить его содержимое в рабочий лист в месте расположения курсора. Эти команды совместно с командами **Copy as Maple Text** и **Paste Maple Text** рассмотрены нами достаточно подробно в предыдущем разделе.

Команда **Undo** (Отменить действие) позволяет отменить действие последней выполненной команды или ввод с клавиатуры. Ее напарник — команда **Redo** — наоборот, позволяет снова выполнить последнее действие, отмененное командой **Undo**. Для этих команд на панели инструментов предусмотрены, соответственно, кнопки и .

Команда **Select All** (Выбрать все) позволяет выделить все содержимое рабочего листа, например, для его копирования в Буфер обмена, а команда **Delete Paragraph** (Удалить абзац) удаляет текстовый абзац, в котором в данный момент расположен курсор.

Команда **Find** в версии Maple 6 изменена таким образом, что позволяет не только осуществлять поиск в рабочем листе заданного текста, но и заменять его на другую текстовую строку, определяемую в диалоговом окне команды.

Команда **Hyperkinks** предназначена для редактирования гиперссылок в рабочем листе на другие рабочие листы, страницы Справки Maple, а также на ресурсы Интернета, задаваемые URL-адресами.

Последняя группа команд меню **Edit** служит для работы с группами вычислений, которые нам уже знакомы, и секциями, специальными конструкциями, служащими для структурирования содержимого рабочего листа, и о которых речь пойдет в следующем разделе, посвященном документированию рабочих листов. Подменю **Split or Join** предоставляет команды для разделения группы вычислений на две в месте расположения курсора (**Split Execution Group**) и соединения группы вычислений, в которой находится курсор, с группой, идущей сразу же за ней (**Join Execution Groups**). Здесь же можно найти аналогичные команды, но применимые к секции, соответственно, **Split Section** или **Join Sections**.

Команды подменю **Execute** вычисляют либо весь рабочий лист (**Worksheet**), либо выделенную в нем часть (**Selection**).

Последнее подменю **Remove Output** позволяет удалять области вывода из выделенной части рабочего листа (команда **Selection**) или во всем рабочем листе (команда **Worksheet**).

Команда **Insert Mode** переключает режим ввода текста в абзаце текстового комментария на вставку в него невыполняемых математических формул и наоборот. Подобные действия реализуются соответствующими кнопками на панели инструментов приложения Maple.

Напоследок мы оставили команду **Object**, которая становится доступной, если на рабочем листе выделен внедренный в него объект OLE 2, причем после имени команды добавляется типовое имя внедренного объекта, например, если внедрен лист MS Excel, то команда выглядит следующим образом: **Object Лист**. Вернее, должна была бы так выглядеть! Однако при работе в русской версии операционной системы Windows 98, Second Edition эта команда всегда остается недоступной, но в списке команд меню **View** появляется команда **Объект** с последующим именем внедренного объекта OLE 2, действие которой идентично действию команды **Edit** ➢ **Object**, как об этой последней команде написано в документации. Действительно, появляющаяся в меню **View** новая команда, вернее подменю, **Объект** позволяет выполнить действия над объектом OLE 2: изменить его или открыть. Список команд этого подменю состоит из двух команд: **Изменить** и **Открыть**, причем обе на русском языке.

1.4.2.3. Команды меню *View*

Команды этого меню позволяют пользователю осуществить настройку интерфейса пользователя системы Maple, а также задать режимы отображения информации на рабочем листе.

Команды первой группы из списка команд являются простыми переключателями. Они включают или выключают режим отображения панели инструментов (команда **Tool Bar**), контекстной панели инструментов (команда **Context Bar**) и строки состояния (команда **Status Line**). Если соответствую-

щий элемент интерфейса отображается, то, как указывалось ранее, слева от имени команды стоит "галочка".

Команды подменю **Zoom Factor** (Изменить масштаб) позволяют изменять масштаб отображения рабочих листов: увеличивать или уменьшать размеры рабочих листов в окне приложения Maple. На основной панели инструментов расположены три кнопки с лупами разных размеров, которые соответствуют масштабам 100%, 150% и 200% отображения рабочих листов.

Команды подменю **Bookmarks** (Закладки) позволяют установить закладку в рабочем листе и быстро переходить к месту, отмеченному определенной закладкой. Это средство удобно для работы с большими документами, а также для отметки, например, текста, который требует редактирования и доработки.

Команда **Show Invisible Characters** (Показать непечатаемые символы) включает или выключает режим отображения непечатаемых символов на рабочем листе. Такими символами являются пробел и символ абзаца (¶). Пробел при таком режиме отображается в виде точки, расположенной выше базовой линии строки (чтобы не спутать с точкой — знаком препинания, расположенной на базовой линии строки). Эта команда полезна для обнаружения ошибки в синтаксисе команды, когда случайно в ее текст вставлен непечатаемый символ, в результате чего синтаксический анализатор Maple выдает ошибки трансляции. Для этой команды на панели инструментов есть кнопка с символом завершения строки ¶.

Команды-переключатели **Show Section Ranges** и **Show Group Ranges** включают и выключают режим отображения квадратных скобок, расположенных слева от группы вычислений и секции и отмечающих их протяженность.

Две последние команды из списка команд меню **View** раскрывают (**Expand All Sections**) или сворачивают (**Collapse All Sections**) все секции в рабочем листе.

Перечисленные команды присутствовали в описываемом меню и в предыдущей версии Maple. В новой версии в него были добавлены новые команды, а команда **Show OLE type** перенесена в него из меню **Edit** (там она называлась **Show OLE Objects**). Эта команда, если включена, позволяет отличить внедренный объект OLE от связанного. Если объект внедрен, то он находится в документе Maple, и любые изменения с его источником не отражаются на нем; если объект OLE связан с источником, то изменения в источнике отражаются и на объекте в рабочем листе Maple. При включенной команде **Show OLE type** внедренный объект отображается в контуре из сплошной линии, а связанный — в контуре из линии, состоящей из точек.

Новые команды **Back** и **Forward** становятся доступны, если были осуществлены переходы по гиперссылкам к другим рабочим листам. Первая команда позволяет вернуться на одну гиперссылку назад, а вторая команда, наоборот, обеспечивает перемещение вперед по гиперссылкам. Их действие полностью соответствует подобным же командам в любом обозревателе Интер-

нета и дублируется, соответственно, кнопками и на панели инструментов.

Если в предыдущих версиях Maple, чтобы уменьшить размеры рабочего листа при работе с ним в окне приложения можно было только удалить из рабочего листа области вывода, то в новой версии Maple 6 можно просто скрыть, а потом обратно отобразить вставленные электронные таблицы, всю область ввода или вывода, а также выведенную в рабочий лист графику. Эти действия обеспечивают команды-переключатели нового подменю **Hide Content**.

В меню **View** также содержится подменю **Palettes**, которое было подробно рассмотрено в предыдущем разделе.

1.4.2.4. Меню *Insert*, *Format*, *Options* и *Window*

Команды меню **Insert** (Вставить) предназначены для работы с рабочими листами. Первая группа команд позволяет переключаться между вставкой в них текстовых комментариев (**Text**), не вычисляемых математических выражений (**Standard Math**), команд, записанных как в синтаксисе Maple (**Maple Input**), так и в привычной математической нотации (**Standard Math Input**). Все эти команды подробно разбирались в предыдущих разделах данной главы.

Большая группа команд позволяет вставить в рабочий лист специальные объекты. Команды подменю **Execution Group** вставляют новую группу вычислений перед курсором (**Before Cursor**) и после курсора (**After Cursor**). Заметим, что для команды вставки группы вычислений после курсора на панели инструментов есть соответствующая кнопка . Действие команд подменю **Paragraph** аналогично действию команд предыдущего подменю, только они вставляют область ввода текстового комментария до (**Before**) и после (**After**) курсора.

Структурирование содержимого рабочего листа в Maple осуществляется с помощью специальных объектов "секция" и "подсекция", для добавления которых в рабочий лист служат команды **Section** и **Subsection**. О том, как они работают, мы расскажем в следующем разделе в связи с вопросами документирования рабочих листов.

Команда **Spreadsheet** нам уже знакома. Ее действие заключается во вставке электронной таблицы Maple в рабочий лист.

Интересную возможность для тех, кто использует Maple в интерактивном режиме при решении своих задач, предоставляет подменю **Plot**. Его две команды **2D** и **3D** вставляют в рабочий лист, соответственно, пустой двумерный и трехмерный график, по сути, просто оси координат, но эти пустые графики обладают такой же функциональностью, как и графики, построенные с помощью функции `smartplot()` контекстного меню области вывода. Если на рабочем листе в области вывода выделить определение

функции и перетащить его мышью в построенные командами **2D** или **3D** пустые графики, то в них будет построен график этой функции. Понятно, что в двумерный пустой график следует перетаскивать функцию одной переменной, а в трехмерный — двух переменных. На подобное поведение графиков указывает надпись `Live` в правом верхнем углу их области вывода.

Замечание

На одном таком графике можно отобразить несколько функций, последовательно перетаскивая их мышью. Технология удаления графиков аналогична такой же для графиков, построенных функцией `smartplot()`. Контекстное меню позволяет изменить параметры построенного графика.

Команда **Hyperlinks** вставляет в рабочий лист гиперссылки на другие рабочие листы, страницы справки Maple или ресурсы Интернета, представленные своими URL-адресами, ссылки на которые задаются в диалоговом окне этой команды.

В Maple 6 добавлена возможность разбиения рабочего листа на страницы. Новая команда **Page Break** как раз и служит этой цели. Она завершает одну страницу и начинает новую, вставляя в рабочий лист разделитель страниц следующего вида:

```
---------------------------------- Page Break ----------------------------
```

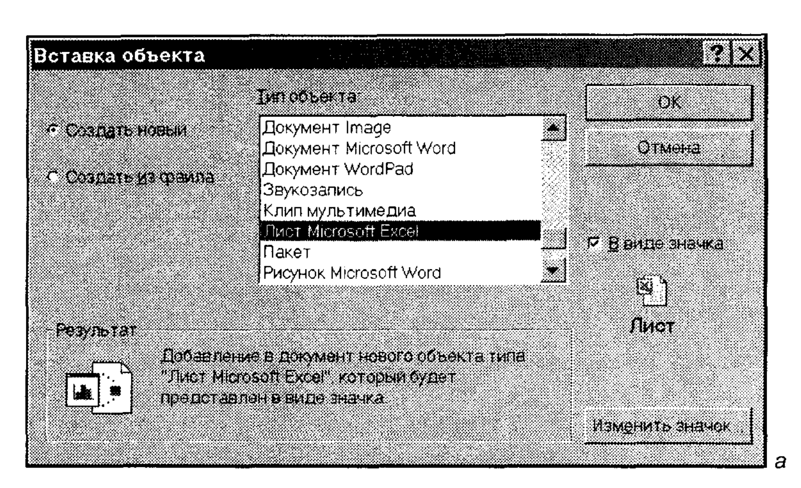
Последняя, не рассмотренная нами команда этого меню **OLE Object** вставляет в рабочий лист объекты OLE 2, созданные другими приложениями Windows, поддерживающими эту технологию. Данная команда отображает стандартное диалоговое окно **Вставка объекта** (русские названия диалоговых окон и объектов на них связаны с используемой автором русской версией Windows 98, у пользователей других версий Windows могут быть отличия в названиях), позволяющее задать в списке **Тип объекта** объект, который предполагается вставить в рабочий лист. При этом можно создать новый объект, запустив соответствующее приложение (рис. 1.23, *а*), или вставить объект, сохраненный в файле (рис. 1.23, *б*).

Для вставки объекта из файла в диалоговом окне следует установить переключатель **Создать из файла** и в поле ввода **Файл**, появившемся в диалоговом окне, задать полное имя файла, либо, нажав кнопку **Обзор**, указать нужный файл. При этом объект можно встроить в рабочий лист или связать с объектом-источником. В этом случае любые изменения в объекте-источнике будут отображаться и в объекте OLE, расположенном в рабочем листе.

Для создания нового объекта следует установить переключатель **Создать новый** и выбрать нужной объект из списка **Тип объекта** диалогового окна. При нажатии кнопки **OK** загрузится приложение, соответствующее типу вставляемого объекта, в котором следует создать новый объект.

Меню **Format** связано с заданием разнообразных стилей вывода информации. Для новичков это не очень существенно, так как обыкновенно начи-

нающего пользователя вполне устраивают системные установки стилей отображения текста в области ввода и вывода. Мы немного расскажем о командах этого меню в разделе, посвященном документированию рабочих листов.



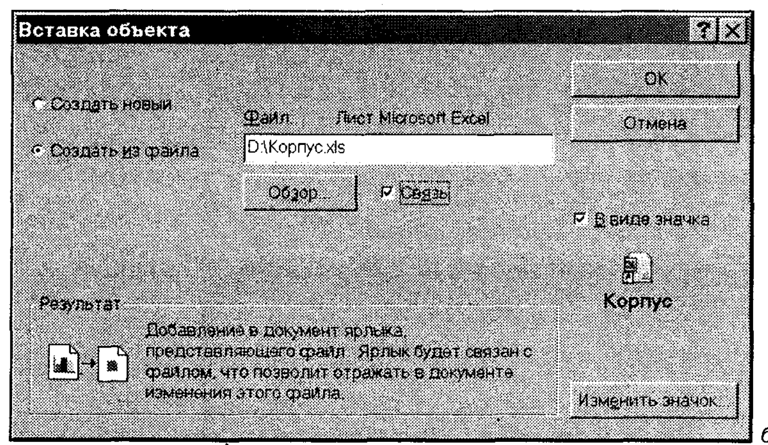


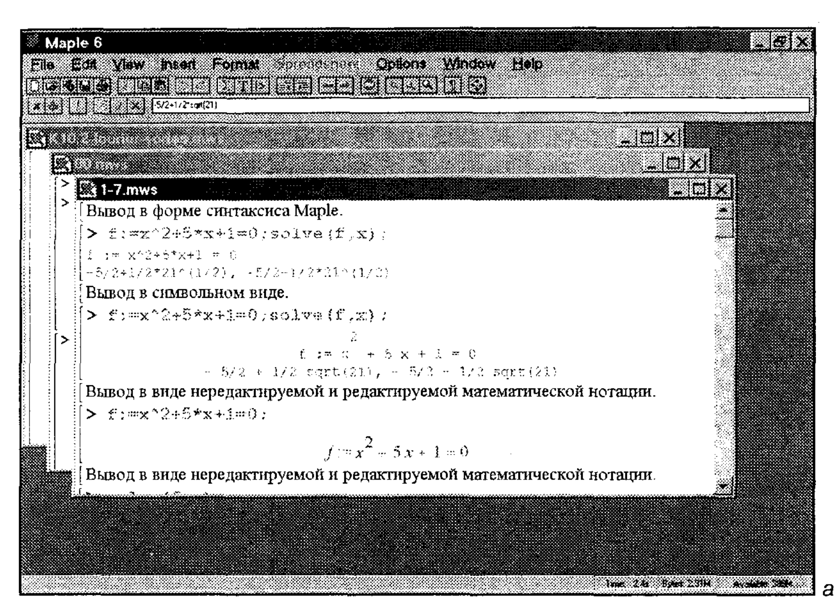
Рис. 1.23. Диалоговые окна вставки объекта OLE

Команды меню **Window** предназначены для работы с несколькими открытыми документами Maple. Они позволяют задать расположение окон документов в окне интерфейса Maple. Их можно расположить каскадом (команда **Cascade**, рис. 1.24, *а*), плиткой (команда **Tile**, рис. 1.24 *б*), горизонтально (команда **Horizontal**, рис. 1.24, *в*) и вертикально (команда **Vertical**, рис. 1.24, *г*).

С помощью команды **Arrange Icons** (Выстроить значки) упорядочиваются значки свернутых окон документов.

Две команды позволяют закрыть все открытые окна (**Close All**) или же закрыть окна, содержащие справочную информацию (**Close All Help**).

В этом разделе мы кратко, но достаточно полно описали все команды стандартного меню рабочего листа. Многие из них повторяются в меню для работы с графикой. Единственное меню, которое мы не затронули в этом разделе, — это меню **Help**, но ему будет посвящен целый раздел данной главы.



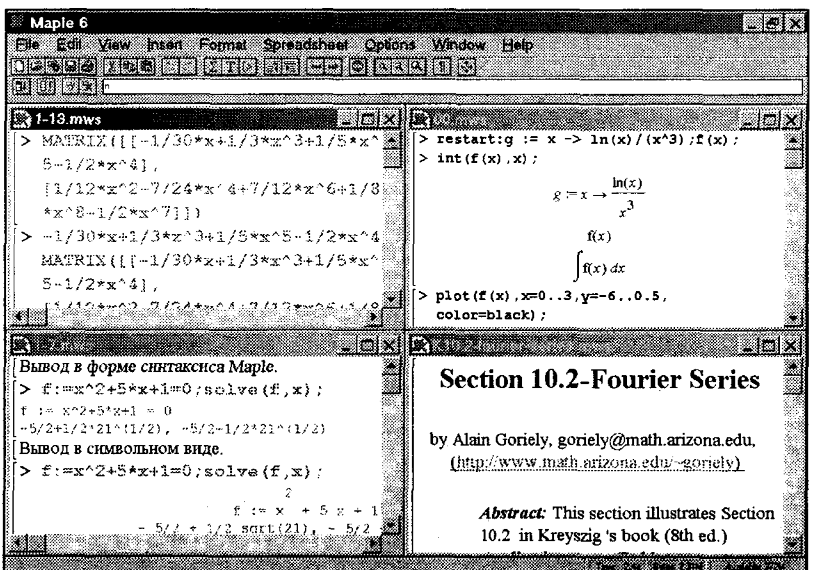
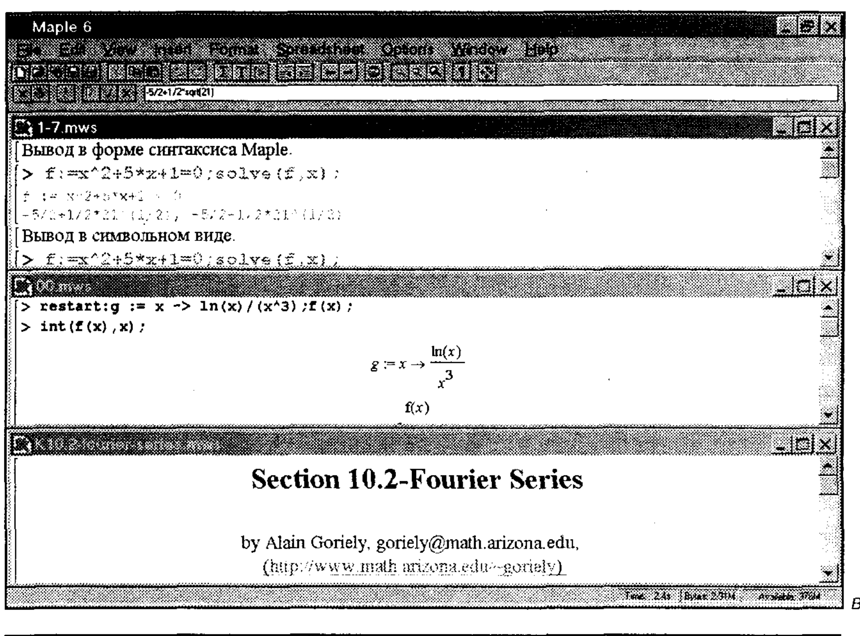


Рис. 1.24. Различные способы расположения документов в окне интерфейса Maple *(см. продолжение)*



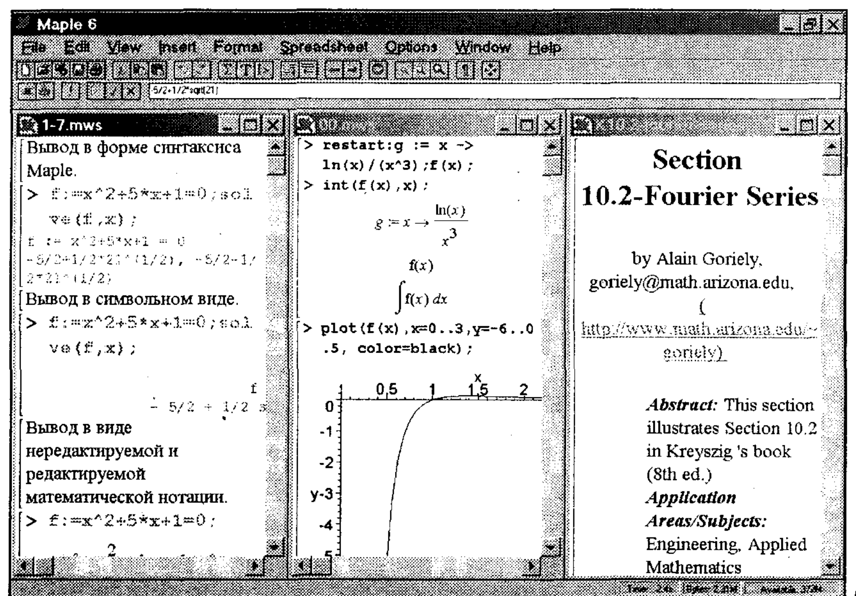


Рис. 1.24. *Окончание*

Кроме этих команд в меню **Window** отображаются все открытые в данном сеансе документы, что позволяет легко и просто переходить от одного документа к другому.

1.5. Документирование рабочих листов

До сих пор наша работа в Maple происходила следующим образом: если нам нужно было что-то вычислить, мы в очередной группе вычислений вводили подходящую команду, получали результат и при необходимости отображали его в графическом виде. Никаких комментариев мы практически не создавали, так как все делали сами и понимали, что у нас есть на рабочем листе. А если необходимо передать результаты нашей работы заказчику или создать учебно-методический материал? Естественно, в таком виде наш рабочий лист совершенно не пригоден для этих целей. Этот раздел как раз и посвящен тому, как сделать в Maple документ, пригодный для чтения другими пользователями.

1.5.1. Структурирование документа

Создание структуры документа рассмотрим на примере рабочего листа вычисления интеграла, на котором мы демонстрировали во введении возможности системы Maple.

Итак, вспомним, что мы вычисляли неопределенный интеграл от функции $y=\frac{\ln(x)}{x^3}$. Наш рабочий лист выглядел достаточно просто, но совершенно не понятно для непосвященного:

```
> f := x -> ln(x)/(x^3);
```

$$f:=x\to\frac{\ln(x)}{x^3}$$

```
> int(f(x),x);
```

$$-\frac{1}{2}\frac{\ln(x)}{x^2}-\frac{1}{4}\frac{1}{x^2}$$

Сначала добавим название нашего рабочего листа и автора. Для этого, установив курсор на первой команде Maple, добавим перед ним новую группу вычислений (**Insert ➢ Execution Group ➢ Before Cursor**) и, так как нам необходимо вводить текст, преобразуем ее в текстовый комментарий, нажав кнопку **T** на панели инструментов. Исчезло приглашение Maple, и контекстная панель инструментов теперь приспособлена для ввода текста. Ее самый левый раскрывающийся список содержит все возможные стили форматирования информации на рабочем листе. При вставке текстового комментария стилем по умолчанию является стиль Normal. Выберем стиль Title (Название), введем название нашего рабочего листа и нажмем клавишу <Enter>. В поле стилей появился стиль Author (Автор), и мы можем ввести имя автора рабочего листа. Результат наших действий показан на рис. 1.25.

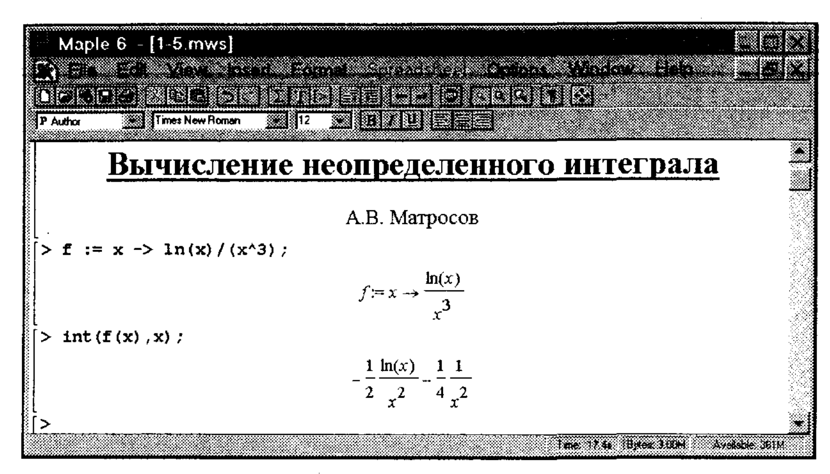


Рис. 1.25. Рабочий лист с заголовком

Замечание

Посмотреть все определения стилей можно командой **Format ➢ Style**. Большинство стилей основывается на каком-то более простом. В диалоговом окне можно задать параметры шрифта стиля: его размеры, начертание, цвет; какой стиль применяется после данного, смещен или нет текст в данном стиле от левой границы окна и т. д.

Теперь заключим наши две группы вычислений в одну секцию. Для этого выделим их и выполним команду **Format ➢ Indent**, которая заключает текущее выделение на рабочем листе в *секцию* — специальный объект рабочего листа, элементами которого являются небольшой серый квадрат со знаком плюс (+) или минус (–) и вертикальная скобка, внутри которой заключены группы вычислений и текстовые комментарии, входящие в данную секцию. Секция может быть раскрыта, как показано на рис. 1.26, и тогда отображается квадрат со знаком минус, вертикальная скобка и все содержимое секции, а может быть закрыта, и тогда отображается только квадрат со знаком плюс, а все содержимое секции будет скрыто. В дальнейшем при ссылке на элемент-квадрат секции будем называть ее кнопкой раскрытия.

После вставки секции установим курсор сразу же после кнопки раскрытия и введем название секции. Если нажать клавишу <Enter>, то автоматически вставится текстовый абзац, в который можно поместить некоторые пояснения, как показано на рис. 1.26.

Теперь вставим комментарий после первой команды, определяющей функцию `f(x)`. Установим курсор в область ее вывода, нажмем кнопку [>] на основной панели инструментов, а затем кнопку [T]. Будет создана область текстового комментария, куда переведем курсор и наберем текст, показанный на рис. 1.27.

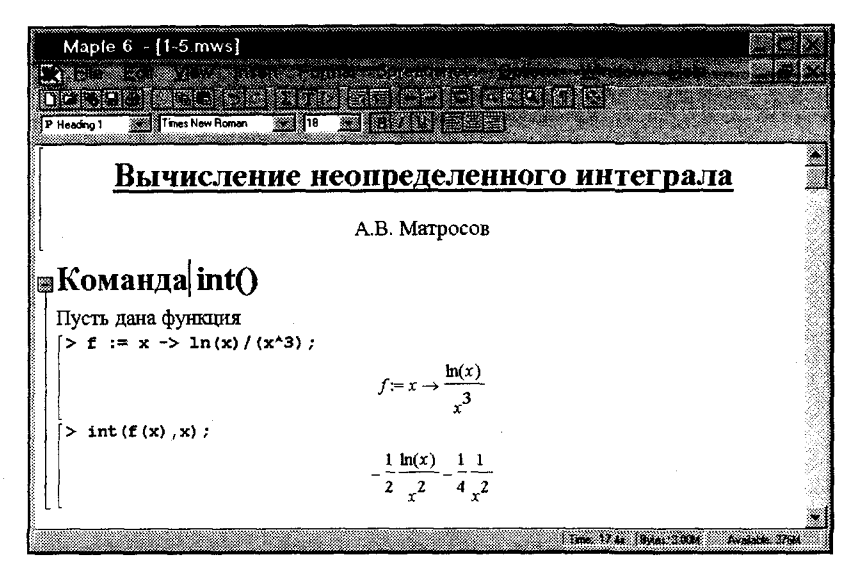


Рис. 1.26. Секция на рабочем листе

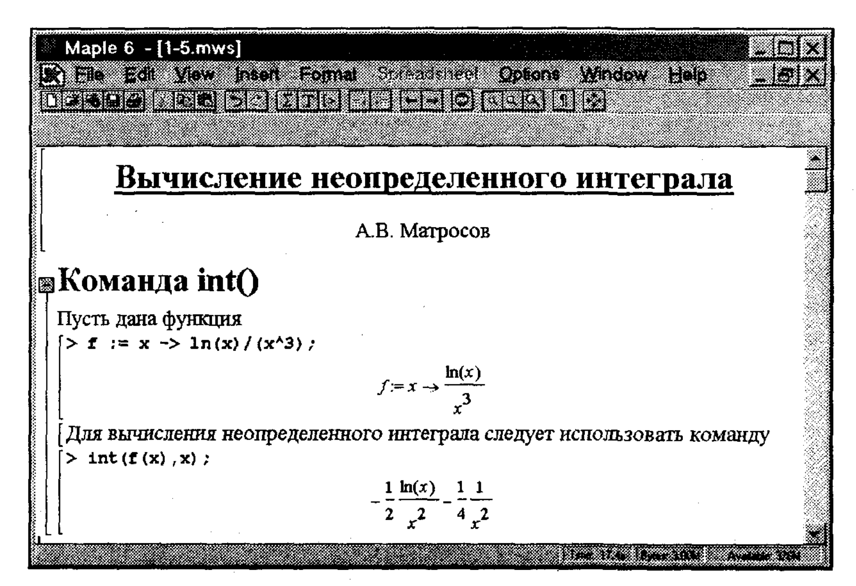


Рис. 1.27. Окончательный вид документа

Наш рабочий лист стал выглядеть намного лучше, однако, задание подынтегральной функции в отдельной группе вычислений смотрится не очень красиво. Если бы ее определение находилось непосредственно в первом текстовом комментарии, то оно смотрелось бы лучше. Вспомним, что Maple позволяет встраивать вычисляемые математические формулы в текст ком-

ментария, и воспользуемся этим. Прежде всего удалим группу определения функции, выделив ее вместе с охватывающей левой скобкой, установим курсор в конец текста первого комментария, выполним команду **Insert ➤ Standard Math Input** и в поле ввода контекстной панели инструментов введем определение функции `f(x)`, она отобразится в тексте комментария в математической нотации. Результат всех наших действий показан на рис. 1.28.

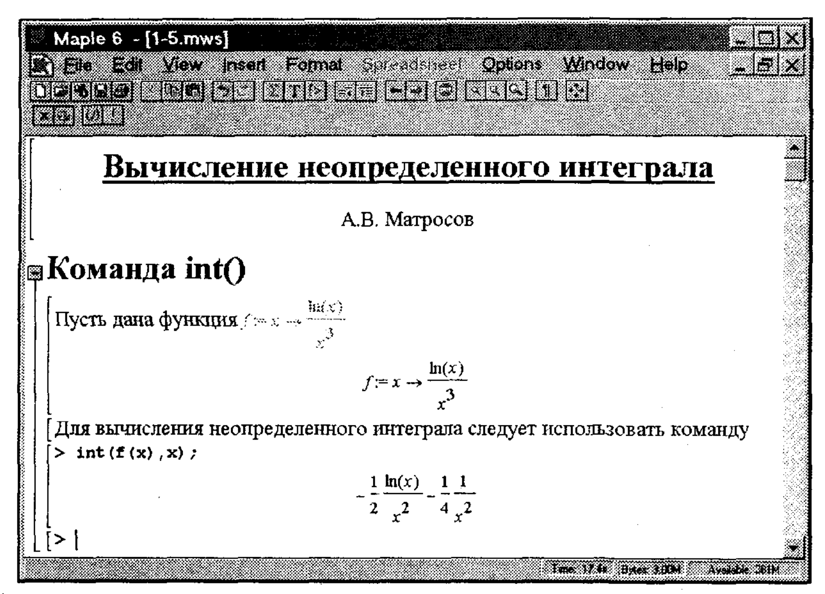
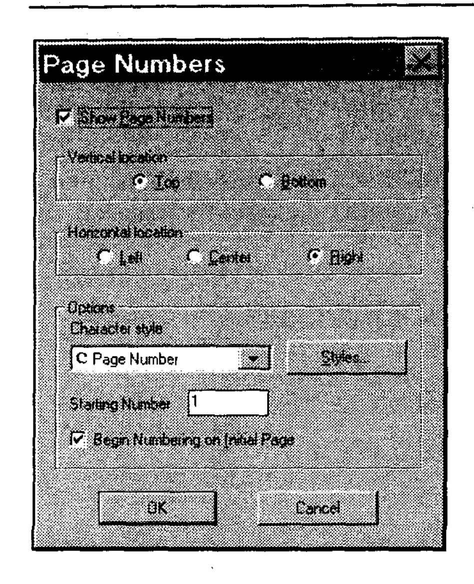


Рис. 1.28. Улучшенный вид документа

Можно теперь продолжить разработку нашего документа, добавив в него секцию, в которой строится график, но мы не будем этого делать, а воспользуемся возможностью Maple работать с несколькими рабочими листами в одном сеансе и построим график полученного интеграла на другом рабочем листе, связав их с помощью гиперссылок.

Но чтобы завершить описание построения документа, мы добавим нумерацию страниц (хотя в нашем документе только одна страница). Для этого в меню **Format** следует вызвать команду **Page Numbers** и в появившемся диалоговом окне установить флажок **Show Page Numbers**, сразу же станут доступными все другие элементы управления этого диалогового окна (рис. 1.29), позволяющие установить параметры для номеров страниц.

В группе переключателей **Vertical location** следует выбрать один из двух переключателей, указывающих наверху страницы (**Top**) или внизу (**Bottom**) будет располагаться ее номер. В группе **Horizontal location** устанавливается расположение номера по ширине страницы: **Left** — слева, **Center** — по центру



и **Right** — справа. В группе **Options** можно выбрать стиль текста для номеров страниц, задать начальный номер страницы в поле **Starting Number**, а также печатать ли номер на первой странице рабочего листа — флажок **Begin Numbering on Initial Page**.

Рис. 1.29. Диалоговое окно установки нумерации страниц рабочего листа

Внимание!

При работе в Maple номера страниц не отображаются, но они действительно печатаются при выводе документа на печать.

Завершая разговор о форматировании рабочих листов следует сказать, что команда **Outdent** меню **Format** удаляет секцию для текущего выделения на рабочем листе. Команды этого же меню **Italic**, **Bold** и **Underline** предназначены для форматирования выделенного текста в комментариях соответственно курсивным, полужирным и подчеркнутым шрифтами. Эти три команды работают как переключатели, поэтому возможно их комбинированное использование для одного и того же текста.

1.5.2. Работа с несколькими рабочими листами

Итак, снова вернемся к нашему рабочему листу вычисления интеграла и сохраним его под каким-либо именем, например 1-5.mws (это нам потребуется в дальнейшем для создания ссылки на него из другого рабочего листа). Теперь создадим новый рабочий лист, нажав кнопку основной панели команд или выполнив команду **File** ➢ **New**. Командой **Windows** ➢ **Vertical** зададим режим отображения всех открытых рабочих листов по ширине окна рабочего поля приложения Windows. Так как у нас открыты два документа, то именно их мы и увидим, причем один должен быть пустым. Для быстрого заполнения рабочего листа мы используем технологию перетаскивания мышью между рабочими листами. Выделим содержимое рабочего листа вычисления интеграла и перетащим его мышью в пустой рабочий лист. Мы это сделали не только в учебных целях, но и для того, чтобы не делать зано-

во структурирование и форматирование рабочего листа. Развернем новый рабочий лист на всю рабочую область и откорректируем его так, как показано на рис. 1.30.

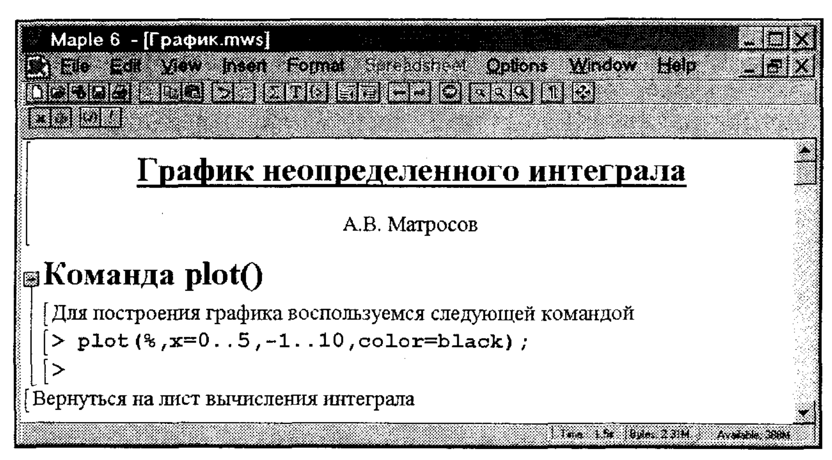


Рис. 1.30. Содержимое рабочего листа (График.mws)

Его структура такая же, как и документа 1-5.mws, но после секции добавлен текстовый комментарий. Именно этот рабочий лист мы будем загружать с помощью гиперссылки. Но сначала мы создадим гиперссылку для возврата на наш начальный лист 1-5.mws. Для этого выделим слово "лист" в последнем текстовом комментарии и выполним команду **Format ➤ Convert to ➤ Hyperlink**. Обратите внимание, что изменился внешний вид этого слова и отобразилось диалоговое окно, в котором следует указать имя рабочего листа 1-5.mws и его расположение, предварительно установив переключатель **Worksheet**. Теперь, при расположении курсора мыши над этой гиперссылкой он меняет вид на изображение руки, тем самым говоря пользователю, что при щелчке на этом слове загрузится другой документ. Вставка гиперссылок на другие ресурсы ничем не отличается от вставки ссылки на рабочие листы.

Сделаем еще один последний шаг в оформлении этого рабочего листа — создадим в нем закладку, на которую автоматически встанет курсор при переходе в этот документ. Таким естественным местом в данном рабочем листе является команда отображения графика функции. Установим курсор в любое ее место и выполним команду **View ➤ Bookmarks ➤ Edit Bookmark**. В появившемся диалоговом окне в поле **Bookmark Text** зададим имя закладки "график". После этого сохраним созданный документ под именем График.mws.

Вернемся к документу 1-5.mws. Добавим в него после секции **Команда int()** область текстового комментария, в котором наберем фразу: "Построение вычисленного интеграла". Установим курсор между первым и вторым сло-

вом и выполним команду **Insert ➤ Hyperlink**. В появившемся диалоговом окне в поле **Link Text** введем слово "график", установим переключатель **Worksheet**. С помощью кнопки **Browser** зададим ссылку на файл График.mws и в раскрывающемся списке **Bookmark** выберем пункт "график". Нажатие кнопки **OK** приведет к появлению в тексте гиперссылки *график*. Теперь щелчок на ней приведет к загрузке и отображению в окне приложения Maple файла График.mws, причем курсор будет находиться в поле команды черчения, и нам остается только нажать <Enter> для построения графика искомой функции. По ссылке *лист* мы можем вернуться на исходный рабочий лист.

Теперь необходимо сделать несколько пояснений. Сначала относительно команды построения графика интеграла в рабочем листе График.mws. Она имеет следующий вид:

```
> plot(%,x=0..5,-1..10,color=black);
```

В ней первым параметром представлена ссылка на результат выполнения последней команды, а таковой являлось вычисление интеграла на листе 1-5.mws. Здесь возникает естественный вопрос: "Неужели можно пользоваться вычислениями, выполненными в другом рабочем листе?" Да, можно. По умолчанию программа Maple загружается в режиме совместного использования рабочими листами результатов своих вычислений, т. е. все вычисления, присваивания переменным и т. п., на одном рабочем листе доступны для всех других, открытых в этом же сеансе. Если пользователя не устраивает такое поведение Maple, то он может загрузить его в параллельном режиме, при котором вычисления на каждом листе производятся независимо от вычислений на других рабочих листах. Для этого следует выбрать команду **Parallel Server Maple 6** в группе **Maple 6** меню **Пуск ➤ Программы** операционной системы Windows.

Замечание

При работе с созданными нами только что рабочими листами следует сначала *обязательно* выполнить все команды рабочего листа 1-5.mws, а потом переходить по гиперссылке на лист График.mws, так как он будет использовать результат последней выполненной команды в сеансе Maple. Конечно, можно было присвоить результат выполнения интеграла некоторой переменной и использовать ее при построении графика, но мы сделали так, как сделали.

1.6. Справочная система

Последним меню в списке стандартного меню рабочего листа является меню **Help** (Справка). Справка очень полезное и удобное средство. Она позволяет достаточно быстро найти нужную информацию, не прибегая к разнообразным книгам и пособиям, тем более, что в них обыкновенно отражена не вся информация о системе. Поэтому остановимся подробнее на получении справочной информации в системе аналитических вычислений Maple.

В Maple 6 справочная система отличается от справки в других приложениях, работающих под управлением системы Windows и с точки зрения автора является более удобной, чем получившая наибольшее распространение справка a la Microsoft.

1.6.1. Организация справки

В основу справочной системы положено понятие гипертекста, т. е. документа, в котором имеются ссылки на другой документ. Таким образом, продвигаясь по этим ссылкам можно получать дополнительную информацию, уточняющую или дополняющую информацию, содержащуюся в исходном документе.

Это очень похоже на чтение книги, в которой имеются ссылки на другие главы, разделы или формулы той же книги, или даже на другую книгу. Только при чтении книги необходимо ее перелистывать и обращаться, например, к странице, на которую ссылаются в тексте, или вообще брать в библиотеке другую книгу.

В случае гипертекстовых документов не надо идти в библиотеку, а достаточно щелкнуть кнопкой мыши на соответствующей связи, и необходимый документ отобразится в окне интерфейса Maple.

Такие слова-ссылки (гиперссылки) в гипертекстовых документах отображаются другим цветом и могут быть подчеркнуты. В системе Справки Maple эти ссылки отображаются бирюзовым цветом и подчеркнуты. Если установить курсор мыши на такую связь и щелкнуть кнопкой мыши, то загрузится документ справочной системы, с которым установлена связь через словоссылку.

С помощью гиперссылок можно "путешествовать" по справочной системе Maple, добираясь до нужной информации. Такие "путешествия" полезны еще и тем, что можно натолкнуться на описание каких-то возможностей, с которыми пользователь еще не сталкивался, но которые могут оказаться полезными для него при решении задач проблемной области, в которой он работает.

Гиперссылки есть во многих справочных системах, но вот структура содержания Справки Maple существенно отличается от других. Но, как говорится, "лучше один раз увидеть, чем десять раз услышать". На рис. 1.31 представлен вид документа справочной системы, отображенного в окне приложения Maple.

Ниже меню справки, основной и контекстной панелей инструментов горизонтально располагается содержание справки, имеющее пять уровней вложенности. Если на каком-то уровне информация не помещается в отведенном пространстве, то отображается вертикальная полоса прокрутки. По горизонтали мы можем видеть весь путь к документу в структуре справки:

3 Зак. 663

вышележащие документы подсвечены синим фоном (на нашем рисунке черным). Если на каком-то уровне документ имеет подчиненные, то в конце имени этого документа стоят три точки. Щелчок на любом документе любого уровня приводит к отображению либо имен подчиненных документов в области содержания, либо самого документа, если у него нет подчиненных, в области, расположенной ниже содержания.

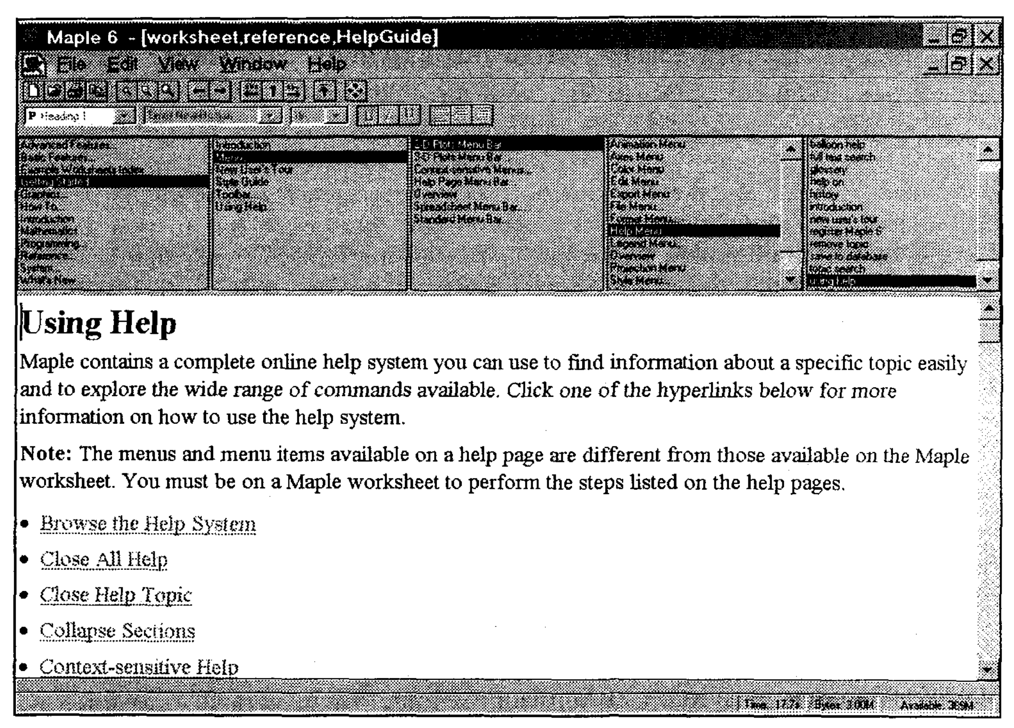


Рис. 1.31. Вид справочной системы Maple

Каждый документ имеет большое количество ссылок на документы, содержание которых тем или иным образом связано с содержанием просматриваемого документа. Щелчок на любой гиперссылке приводит к отображению связанного с ней документа. Автоматически перестраивается и содержание, показывая расположение отображенного документа в структуре содержания.

1.6.2. Вызов справки с помощью меню *Help*

Команда **Introduction** загружает в рабочее окно документ, содержащий введение в систему аналитических вычислений Maple. При этом изменяется список меню в строке меню, панель инструментов, а контекстная панель

инструментов становится недоступной. Документ Справки внешне представляет собой рабочий лист, который нельзя модифицировать, но который содержит гиперсвязи, позволяющие загружать другие документы Справки. Панель инструментов справочной системы содержит кнопки, ассоциированные с командами справочной системы. На рис. 1.32 показана эта панель инструментов с описанием назначения кнопок.

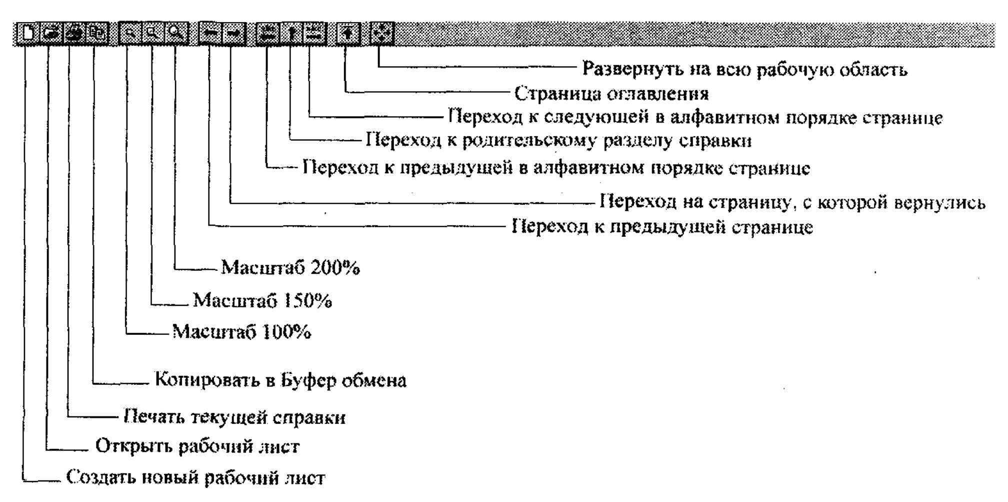


Рис. 1.32. Панель инструментов справочной системы Maple

В меню **Edit** справочной системы есть очень полезная команда — **Copy Examples**. Практически на каждой странице справки есть несколько примеров по использованию описываемых функций. Эта команда позволяет скопировать в Буфер обмена примеры, вставить их в рабочий лист и выполнить. Мы настоятельно рекомендуем проделывать эту процедуру для каждой, вновь изучаемой функции и анализировать получаемые результаты.

Команда **Topic Search** (Поиск раздела) меню **Help** позволяет найти необходимый раздел справки, причем реализована возможность поиска по неполному заданию имени раздела. Это означает, что если в поле **Topic** диалогового окна команды поиска раздела задать несколько первых букв, то в списке **Matching Topics** отобразятся все разделы, начинающиеся с введенной последовательности букв, причем этот поиск будет осуществлен автоматически, если установлен флажок **Auto-search**.

Команда **Full Text Search** (Полный поиск текста) позволяет найти все страницы Справки, в которых присутствует заданный в поле **Word(s)** диалогового окна этой команды текст. В отличие от предыдущей команды для начала поиска здесь необходимо нажать кнопку **Search**. Однако эта команда поиска реализует другую полезную функциональность: рядом с именами найденных страниц Справки в скобках отображается коэффициент соответствия задан-

ных слов поиска и найденной страницы (рис. 1.33). Его значение, равное 1, говорит о полном соответствии страницы заданному запросу. Если оно близко к нулю, то на этой странице встречаются слова из запроса, но в целом страница посвящена другой теме. Эту функциональность можно отключить, сбросив флажок **Goodness of Match**.

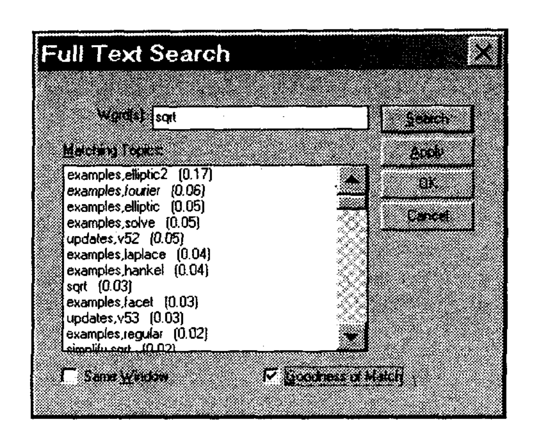


Рис. 1.33. Диалоговое окно команды **Full Text Search**

Полезная команда **History** (История) отображает все просмотренные в данном сеансе страницы Справки и позволяет перейти на любую из них.

При работе со Справкой может быть открыто много справочных страниц. Чтобы закрыть все окна страниц Справки, необходимо в меню **Window** выбрать команду **Close All Help** (Закрыть все страницы Справки).

Команда-переключатель **Balloon Help** очень полезна для начинающих работать с системой Maple. Она включает режим отображения всплывающей подсказки при наведении указателя мыши на элементы основного или контекстного меню, и кнопки основной и контекстных панелей инструментов.

1.6.3. Вызов справки из рабочего листа

Получать справочную информацию можно и не используя команды меню **Help**, а набрав команду `?topic`, где `topic` означает имя пакета (библиотеки), команды или служебного слова Maple. (Заметим, что эта команда не завершается точкой с запятой или двоеточием.) Для получения информации о команде `command` из пакета `topic` можно воспользоваться следующими эквивалентными формами команды получения справки: `?topic,command;` и `?topic[command];`. Обратим внимание на то, что эти команды завершаются разделителем (`;`).

Вместо команд со знаком вопроса в начале можно использовать следующие формы команды `help`, завершающиеся точкой с запятой:

```
help(topic);
help(topic, command);
help(topic[command]);
```

Все перечисленные вызовы справки требуют задания команд в области ввода, что может оказаться неудобным. Система Maple предоставляет более удобный способ получения справки по команде, заданной в рабочем листе. Для этого необходимо поместить курсор на имени команды или служебного слова и нажать клавишу <F1>. Такой способ, конечно, более удобен, если надо быстро получить информацию, например, о параметрах нужной команды или об альтернативных формах ее задания. При этом, естественно, необходимо знать имя команды, поэтому предыдущие способы или обращение к командам меню **Help** остаются единственными способами для определения, например, всех команд какого-нибудь пакета.

ГЛАВА 2



Основные объекты и команды

Система аналитических вычислений Maple является интерактивной системой. В данном случае это означает, что пользователь вводит команду или оператор языка Maple в области ввода рабочего листа и, нажав клавишу <Enter>, сразу же передает ее аналитическому анализатору системы, который выполняет ее. Если команда введена правильно, то в области вывода появляется результат ее выполнения, если она содержит синтаксические ошибки или ошибки выполнения, система печатает сообщение об этом. Для исправления ошибки следует вернуться к оператору, откорректировать его и снова выполнить. После выполнения введенной команды система ожидает очередной команды от пользователя. В любой момент можно вернуться к любой команде или оператору на рабочем листе, подкорректировать его и снова выполнить, причем, если на рабочем листе есть команда, использующая результат вновь вычисленной, то ее следует также снова вычислить, установив на нее курсор и нажав клавишу <Enter>, а если таких команд много, то можно выполнить команду графического интерфейса **Edit ➢ Execute ➢ Worksheet** для повторного вычисления всех команд рабочего листа.

Каждый оператор или команда должны *обязательно* завершаться *разделительным знаком*. Таких в Maple два — точка с запятой (;) и двоеточие (:). Если предложение завершается точкой с запятой, то оно вычисляется, а в области вывода отображается результат. При использовании двоеточия в качестве разделителя команда выполняется, но результаты ее работы не отображаются в области вывода рабочего листа. Это удобно при программировании в Maple, когда нет необходимости в выводе каких-то промежуточных результатов, получаемых из операторов цикла, так как вывод этих результатов может занять много места на рабочем листе, да и может потребоваться значительное количество времени на их отображение.

Внимание!

На протяжении всей книги для команд Maple используется запись в форме синтаксиса языка Maple. Если читатель при выполнении примеров желает отобра-

жать команды в математической нотации, то следует командой Options ➤ Input Display ➤ Standard Math Notation установить соответствующий режим отображения.

2.1. Объекты, переменные и выражения

Как и в любой интерактивной системе, в Maple реализован свой язык, с помощью которого происходит общение пользователя с системой. Базовыми понятиями являются объекты и переменные, из которых с помощью допустимых математических операций составляются выражения.

Простейшими *объектами*, с которыми может работать Maple, являются числа, константы и строки. Если читатель знаком с каким-либо языком программирования, то эти объекты соответствуют литеральным (буквальным) константам разных типов данных в языках программирования.

2.1.1. Числа

Числа могут быть целыми, обыкновенными дробями, радикалами, числами с плавающей точкой и комплексными. Первые три типа чисел позволяют выполнять точные вычисления, без округлений, разнообразных математических выражений, реализуя точную арифметику, что отличает все системы аналитических вычислений, в том числе и Maple, от систем, предлагающих численные решения математических задач, например, MathCad и Matlab. Числа с плавающей точкой являются приближенными, в которых число значащих цифр ограничено. Эти числа служат для приближения, или аппроксимации, точных чисел Maple. Комплексные числа могут быть как точными, если действительная и мнимая части представлены точными числами, так и приближенными, если при задании действительной и мнимой частей комплексного числа используются числа с плавающей точкой.

Целые числа задаются в виде последовательности цифр от 0 до 9. Отрицательные числа задаются со знаком минус (-) перед числом, нули перед первой ненулевой цифрой являются не значащими и не влияют на величину целого числа. Maple может работать с целыми числами произвольной величины, количество цифр практически ограничено числом 2^{28}. Вычисления с целыми числами достаточны просты и реализуют четыре арифметических действия (сложение +, вычитание -, умножение *, деление /) и вычисление факториала (!):

Пример 2.1. Операции с целыми числами

```
> 44+6;
```

```
> 7-10;
```

-3

```
> 6*7;
```

42

```
> 45/5;
```

9

```
> 70!;
```

11978571669969891796072168909873645893814254642585755536286462800958 2\
7898453196800000000000000000

```
> length(%);
```

101

Обратите внимание на то, как Maple представляет большое целое число, которое не помещается в строке области вывода: он использует символ обратного слэша (\) в качестве символа продолжения вывода на следующей строке. Последняя команда вычисляет количество цифр в результате предыдущего вычисления. В ней в качестве параметра используется операция `%`, которая является всего лишь удобной формой ссылки на результат выполнения предыдущей операции. В Maple есть еще две подобные операции, которые идентифицируют результаты предпредыдущей и предпредпредыдущей команд. Их синтаксис выглядит, соответственно, следующим образом: `%%` и `%%%`.

Кроме обычных арифметических действий над целыми числами, Maple предлагает достаточно большой набор команд, позволяющих выполнить действия, специфичные при обработке целых чисел: разложение на простые множители (`ifactor`), вычисление частного (`iquo`) и остатка (`irem`) при выполнении операции целого деления, нахождения наибольшего общего делителя двух целых чисел (`igcd`), выполнение проверки, является ли целое число простым (`isprime`) и многое другое. Примеры использования перечисленных команд приводятся ниже:

Пример 2.2. Команды для работы с целыми числами

```
> ifactor(54);
```

$(2)\,(3)^3$

```
> iquo(45,7);
```

6

```
> irem(45,7);
```

3

```
> 7*%%+%;
```

45

```
> igcd(678,456);
```

$$6$$

```
> isprime(5678945691);
```

$$false$$

В этом примере для проверки вычисления частного и остатка двух целых чисел использованы операции получения результата выполнения предыдущей (вычисление частного) и предпредыдущей (вычисление остатка) команд. Результатом команды `isprime()` является булева константа `true` (истина) или `false` (ложь).

Замечание

Получить список всех команд для работы с целыми числами можно, набрав в области ввода рабочего листа команду `?integer`.

Обыкновенные дроби задаются с помощью операции деления двух *целых* чисел. Заметим, что Maple автоматически производит операцию сокращения дробей. Над обыкновенными дробями можно выполнять все основные арифметические операции.

Пример 2.3. Задание обыкновенных дробей и выполнение действий над ними

```
> 64/24;
```

$$\frac{8}{3}$$

```
> 5/8+4/7;
```

$$\frac{67}{56}$$

```
> 2+9/5;
```

$$\frac{19}{5}$$

```
> 4+8/2;
```

$$8$$

Обратите внимание на последнее вычисление в примере. Если при задании дроби ее знаменатель сокращается, то такая "дробь" трактуется программой Maple как целое число.

Иногда представление результата в виде обыкновенной дроби не совсем удобно, и возникает задача преобразования ее в десятичную дробь. Решить поставленную проблему помогает команда `evalf()`, которая аппроксимирует обыкновенную дробь числами с плавающей точкой, используя десять значащих цифр в мантиссе их представления. Если точность по умолчанию не достаточна, то ее можно задать вторым параметром указанной функции.

Пример 2.4. Преобразование обыкновенной дроби в десятичную

```
> evalf(456/789);
```

$$.5779467681$$

```
> evalf(456/789, 30);
```

$$.577946768060836501901140684411$$

Внимание!

Дробь и ее десятичное представление не являются идентичными объектами Maple. Десятичное представление всего лишь *аппроксимация* точной величины, представленной обыкновенной дробью.

Радикалы задаются как результат возведения в дробную степень целых или дробных чисел, или вычисления из них же квадратного корня функцией `sqrt()`, или вычисления корня *n*-ой степени с помощью функции `surd(число,n)`. В Maple операция возведения в степень задается символом ^ или последовательностью из двух звездочек (**). При возведении в степень дробей их следует заключать в круглые скобки, как, впрочем, и дробный показатель степени. При задании радикалов также производятся возможные упрощения, связанные с вынесением из-под знака радикала максимально возможной величины.

Пример 2.5. Задание радикалов

```
> (3/4)^(2/3);
```

$$\frac{1}{4}3^{(2/3)}4^{(1/3)}$$

```
> sqrt(68/9);
```

$$\frac{2}{3}\sqrt{17}$$

```
> surd(75/3,4);
```

$$\sqrt{5}$$

Замечание

На рабочем листе Maple показатели степени в результате вычисления первого радикала представлены в виде $\left(\frac{2}{3}\right)$ и $\left(\frac{1}{3}\right)$. Тот вид, который они имеют в примере 2.5, является результатом вставки в документ Word области вывода Maple через Буфер обмена.

Вычисления с целыми, дробями и радикалами являются *абсолютно точными*, так как при работе с этими типами данных программа Maple не производит никаких округлений в отличие от чисел с плавающей точкой.

Числа с плавающей точкой задаются в виде целой и дробной частей, разделенных десятичной точкой, с предшествующим знаком числа, например, 2.3456, -3.415. Числа с плавающей точкой можно задавать, используя так называемую экспоненциальную форму записи, в которой сразу же после вещественного числа с плавающей точкой или обычного целого, называемого мантиссой, ставится символ e или E, после которого задается целое число со знаком (показатель степени). Такая форма записи вещественного числа означает, что мантиссу следует умножить на десять в степени числа, соответствующего показателю степени, чтобы получить значение числа, записанного в такой экспоненциальной форме. Например, 2.345e4 соответствует числу 23450.0. Таким образом можно представлять очень большие по абсолютному значению числа (показатель степени положительное число) или очень маленькие (показатель степени отрицательное число).

Из чисел можно составлять математические выражения с помощью арифметических операций. Символами арифметических операций в Maple являются символы, перечисленные в табл. 2.1.

Таблица 2.1. *Арифметические операции*

Символ	Операция
+	Сложение
-	Вычитание
*	Умножение
/	Деление
^ или **	Возведение в степень
!	Факториал (применим только к целым неотрицательным числам)

Последовательность выполнения арифметических операций соответствует стандартным правилам старшинства операций в математике: сначала производится возведение в степень, затем умножение и деление, а в конце — сложение и вычитание. Все действия выполняются слева направо. Операция вычисления факториала имеет наибольший приоритет. Для изменения последовательности арифметических операций следует использовать круглые скобки.

Если в выражении все числа являются целыми, дробями или радикалами, то результат представляется также с использованием этих типов данных, но если в выражении присутствует число с плавающей точкой, то результатом вычисления такого "смешанного" выражения будет также число с плавающей точкой, если только в выражении не присутствует радикал. В этом случае радикал вычисляется точно, а коэффициент при нем вычисляется либо

точно, либо в виде числа с плавающей точкой в зависимости от типа сомножителей.

Пример 2.6. Вычисление "смешанных" арифметических выражений

```
> 3^5*0.1;
```

$$24.3$$

```
> 3^5*(1/10);
```

$$\frac{243}{10}$$

```
> 2e1+4/5+sqrt(3)*4/5*0.1+surd(5,3)*28/10;
```

$$20.80000000 + .08000000000\,\sqrt{3} + \frac{14}{5}\,5^{(1/3)}$$

Скажем так, Maple всегда пытается произвести вычисления с абсолютной точностью. Если это не получается, тогда он подключает арифметику с вещественными числами.

Последние числа, с которыми мы познакомим читателя и с которыми умеет работать Maple, — *комплексные числа*. Для мнимой единицы $\sqrt{-1}$ в Maple используется константа `I`. Задание комплексного числа не отличается от его обычного задания в математике:

```
> 2/5+3*I;
```

$$\frac{2}{5} + 3\,I$$

Maple умеет выполнять все арифметические действия над комплексными числами точно так же, как он это делает и с действительными целыми, дробями и с плавающей точкой.

Пример 2.7. Арифметические операции с комплексными числами

```
> (2/5+3*I)+(4+1/2*I);
```

$$\frac{22}{5} + \frac{7}{2}\,I$$

```
> (2/5+3*I)*(4+1/2*I);
```

$$\frac{1}{10} + \frac{61}{5}\,I$$

```
> (2/5+3*I)/(4+1/2*I);
```

$$\frac{62}{325} + \frac{236}{325}\,I$$

```
> (2/5+3*I)/(4+1.0/2*I);
```

$$.1907692308 + .7261538462\,I$$

Обратите внимание на последнее выражение в примере 2.7. Если хотя бы одна из действительных или мнимых частей комплексного числа вычисляется в виде числа с плавающей точкой, то результат также представляется через эти числа.

Для выделения из комплексного числа действительной и мнимой части в Maple предусмотрены две функции: `Re()` для действительной и `Im()` для мнимой части комплексного числа. Вычислить аргумент комплексного числа можно с помощью функции `argument()`, а построить комплексно-сопряженное — функцией `conjugate()`:

```
> Re(5+3*I);
```

$$5$$

```
> conjugate(5+3*I);
```

$$5-3\,I$$

```
> argument(%);
```

$$-\arctan\left(\frac{3}{5}\right)$$

Замечание

Maple умеет работать также со специальными гауссовыми целыми числами, но мы не касаемся этой темы в нашей книге. Те, кто желает познакомиться с пакетом Maple для работы с этими числами, могут выполнить команду `?GaussInt`.

2.1.2. Константы

Кроме чисел, задаваемых пользователем, Maple содержит целый ряд предопределенных *именованных констант* — констант, к значению которых можно обращаться с помощью некоторого имени. Часть этих констант не может быть изменена, а часть можно изменять. Неизменяемые константы представлены в табл. 2.2.

Таблица 2.2. *Неизменяемые константы*

Константа	Значение
`Catalan`	Число, являющееся суммой ряда $\sum_{i=0}^{\infty}\frac{(-1)^i}{(2\,i+1)^2}$, приближенно равно 0.9159655942...
`false`	Значение "ложь" при работе с булевскими переменными
`true`	Значение "истина" при работе с булевскими переменными
`FAIL`	Используется в качестве третьего значения при вычислении функций трехзначной логики

Таблица 2.2 (окончание)

Константа	Значение
`gamma`	Константа Эйлера $\gamma = \lim_{n \to \infty} \sum_{i=1}^{n} \left(\frac{1}{i} - \ln(n) \right) \approx 0.5772156649...$
`Pi`	Число $\pi = 3.141592654...$
`I`	Мнимая единица $\sqrt{-1}$
`infinity`	Бесконечность ∞

Константы, значения которых могут быть переопределены, — это константы, задающие необходимые для работы программы параметры. К наиболее важным можно отнести две константы, которые влияют на точность вычислений: `Digits` и `Order`. Первая константа задает число значащих цифр для операций с числами с плавающей точкой. По умолчанию она имеет значение `10`. Константа `Order` определяет количество членов в разложении функции в ряд Тейлора (по умолчанию установлена равной `6`).

Пример 2.8. Изменение значения константы `Digits`

```
> evalf(Pi);
```

$$3.141592654$$

```
> Digits:=30;
```

$$Digits := 30$$

```
> evalf(Pi);
```

$$3.14159265358979323846264338328$$

Замечание

Посмотреть все константы, определенные в Maple, можно, выполнив команду `?ininame`.

Замечание

Кроме перечисляемых на странице Справки констант все переменные, имена которых начинаются с `_Env`, по умолчанию являются системными константами Maple.

2.1.3. Строки

Кроме чисел Maple позволяет работать со *строкой* — любым набором символов, заключенным в двойные кавычки, например, `"Это пример строки в Maple"`.

Каждый символ в строке представляет самого себя. Длина строки в Maple практически не ограничена и может достигать на 32-битных компьютерах длины в 268 435 439 символов.

Внимание!

При определении строк следует внимательно следить за ограничивающими двойными кавычками и не задавать вместо них одинарные или обратные. Первые определяют режим отложенных вычислений для выражения, а вторые задают символическое имя, которое можно использовать как переменную.

Если необходимо, чтобы в строке присутствовали двойные кавычки, то следует поместить в строку идущие подряд две двойные кавычки или скрыть их основное назначение с помощью символа обратной наклонной черты (\). При этом в области вывода и пара двойных кавычек, и двойные кавычки с предшествующей обратной наклонной чертой отображаются как пара символов \". Однако интерпретатором Maple эта пара символов рассматривается как один символ двойных кавычек, в чем можно убедиться, выполнив команду `length()`, подсчитывающую количество символов в строке:

```
> "ST\"RING""";
                    "ST\"RING\""
> length(%);
                         8
```

Если идут подряд две строки, разделенные символами-разделителями (пробел, табуляция или переход на новую строку), то эти две строки соединяются в одну, причем значение второй без пробела пристраивается в конец первой строки:

```
> "Одна строка" "Вторая строка";
                    "Одна строкаВторая строка"
```

Соединение строк можно осуществить и с помощью операции конкатенации (`||`), или обращением к функции `cat()`:

```
> "Одна строка" || "Вторая строка";
                    "Одна строкаВторая строка"
> cat("Одна строка","Вторая строка");
                    "Одна строкаВторая строка"
```

Забегая вперед, скажем, что строка представляется как одномерный массив, поэтому можно использовать индекс для выделения подстроки из заданной:

```
> "STRING"[4..5];
                         "IN"
```

```
> s:="STRING";
```

$$s := \text{"STRING"}$$

```
> s[4];
```

$$\text{"I"}$$

2.1.4. Переменные, неизвестные и выражения

Мы познакомились, как работать в Maple с числовыми выражениями, но это только позволяет использовать его как некоторый достаточно "умный" калькулятор, не включая всю мощь аналитических вычислений. Первым шагом к освоению всех возможностей Maple является знакомство с переменными, в которых можно хранить вычисленные значения функций и символьных выражений, а также с неизвестными величинами, которые представляют обычные математические неизвестные, когда мы решаем задачу на листке бумаги, и используются для задания символьных выражений Maple.

Каждая *переменная* Maple имеет имя, представляющее последовательность латинских символов, начинающихся с буквы, причем строчные и прописные буквы считаются различными. (Про такие системы говорят, что они чувствительны к регистру.) Кроме букв в именах переменных могут использоваться также цифры и знак подчеркивания, однако первым символом в имени должна быть буква. Примеры различных имен:

```
MyName, myname, my_name
```

Замечание

В именах переменных можно использовать и буквы национального алфавита, в частности русского, однако подобная практика, с нашей точки зрения, кажется не практичной. Во-первых, она требует постоянного переключения клавиатуры с английского языка на русский и наоборот, а во-вторых, в математике все-таки используют латинский и греческий алфавиты.

В качестве имен запрещено использовать зарезервированные слова языка Maple:

```
and          end      in         od       save
break        error    intersect  option   stop
by           export   local      options  then
catch        fi       minus      or       to
description  finally  mod        proc     try
do           for      module     quit     union
done         from     next       read     use
elif         global   not        return   while
else         if
```

Нельзя также использовать так называемые защищенные слова Maple, к которым, в частности, относятся имена неизменяемых констант. Попытка присвоить такому имени какое-либо значение приводит к ошибке:

```
> Catalan:=7;
Error, attempting to assign to `Catalan` which is protected
Ошибка, попытка присвоить значение защищенному символу `Catalan`
```

Замечание

Узнать о защищенных именах можно, отобразив страницу Справки командой `?protect`.

Можно задавать переменные с именами, содержащими пробелы, но для этого их следует заключать в обратные кавычки:

```
> `Name with space`:=789;
```

$$Name\ with\ space := 789$$

```
> `Name with space`;
```

$$789$$

Замечание

Собственно говоря, любое правильное имя также можно заключить в обратные кавычки и от этого ничего страшного не произойдет, так как основное действие обратных кавычек (семантика) заключается в создании символического имени (в Maple этот объект имеет тип `symbol`).

Выражение представляет собой комбинацию имен переменных, чисел и, возможно, других объектов Maple, соединенных знаками допустимых операций (мы пока немного знакомы с арифметическими операциями, другие допустимые операции будут рассмотрены в главе 5, посвященной программированию). Единственным предназначением выражения является его вычисление и получение некоего результата, который можно использовать в операторах языка Maple при дальнейших вычислениях.

Если в выражении используется переменная, которой не присвоено никакого числового или строкового значения, то такая переменная рассматривается системой Maple как некая *неизвестная* величина, а выражение, содержащее неизвестные, называется *символьным выражением*. Именно для работы с такими выражениями прежде всего и разрабатывался Maple:

```
> x^2+5*x+1;
```

$$x^2 + 5\,x + 1$$

```
> sqrt(exp(sin(x*y)));
```

$$\sqrt{e^{\sin(x\,y)}}$$

Обратите внимание, Maple в области вывода действительно печатает неизвестные переменные как простые математические неизвестные, имена которых соответствуют именам переменных.

Для работы с символьными выражениями существует огромное количество функций или команд. Например, можно вычислить производную символьного выражения по какой-либо неизвестной или интеграл, в котором в качестве подынтегральной функции используется символьное выражение, можно упростить его, приведя к виду, удобному для дальнейшего применения и многое другое. Основная деятельность пользователя Maple и направлена на выполнение разнообразных преобразований с символьными выражениями, и, собственно, вся наша книга именно этому и посвящена.

Важной операцией в Maple, связанной с выражениями, является операция *присваивания* (:=). Она имеет следующий синтаксис:

```
переменная := выражение;
```

Здесь в левой части задается имя переменной, а в правой части любое выражение, которое может быть числовым, символьным или просто другой переменной. Семантика (смысл) этого оператора в том, что переменной в левой части присваивается значение выражения, стоящего в правой части. В дальнейшем, если будет необходимо использовать выражение из левой части операции присваивания, то достаточно сослаться на имя переменной, указанное в правой части операции.

Переменные позволяют хранить и обрабатывать разнообразные типы данных, с которыми работает Maple. Мы уже знакомы с такими типами данных, как целый (`integer`), дробь (`fraction`), числовой вещественный с плавающей точкой (`float`) и строка (`string`). Кроме этих типов данных существует еще большое множество типов, необходимых для выполнения аналитических преобразований: функция (`function`), индексные данные (`indexed`), множество (`set`), список (`list`), ряды (`series`), последовательность выражений (`exprseq`) и некоторые другие. Постепенно по мере продвижения нашего изучения Maple мы познакомимся со всеми этими типами данных.

Замечание

Перечисление всех допустимых типов данных Maple представлено в справочной странице, отображаемой командой `?type`.

По умолчанию переменная Maple имеет тип `symbol`, представляющий символьную переменную, и ее значением является ее собственное имя. Поэтому простое объявление переменной `m` оператором `m;` приведет к отображению в области вывода рабочего листа имени этой переменной.

Пример 2.9. Задание и определение типа символьной переменной

```
> m;
```

$$m$$

```
> whattype(m);
```

symbol

В примере 2.9 можно видеть функцию `whattype()`, которая определяет тип выражения или переменной, заданных в качестве ее параметра.

То, что переменная по умолчанию имеет символьный тип, оказывается очень полезным при использовании функций. Если имя функции Maple задано не совсем правильно, или такой функции не существует, или не подключен пакет, где она расположена, то ответом Maple на попытку вычислить ее будет отображение в области вывода не результата выполнения функции, а полностью повторенная строка области ввода.

При присвоении переменной какого-нибудь значения, ее тип изменяется на тип присвоенного ей значения.

Переменные можно использовать для составления выражений наряду с числами. Все, сказанное выше о числовых выражениях и порядке их вычисления, относится и к выражениям, содержащим переменные.

Обычно в математических выражениях используются разнообразные математические функции. В Maple изначально определен большой набор стандартных математических функций, начиная от элементарных и заканчивая специальными функциями, которые используются при решении сложных задач математической физики.

В табл. 2.3 представлены основные математические функции и соответствующий им синтаксис Maple.

***Таблица 2.3.** Основные математические функции*

Функция	Синтаксис Maple	Функция	Синтаксис Maple
e^x	`exp(x)`	$\sqrt{x}$	`sqrt(x)`
$\ln(x)$	`ln(x)` или `log(x)`	$\|x\|$	`abs(x)`
$\log_{10}(x)$	`log10(x)`	$\mathrm{sgn}(x)$	`signum(x)`
$\log_a(x)$	`log[a](x)`	$n!$	`n!`

Тригонометрические и гиперболические функции сведены в табл. 2.4. Обратим внимание на несоответствие записи некоторых функций в русскоязыч-

ной математической литературе и в англоязычной, например функции тангенса угла. Значения параметров тригонометрических функций задаются в радианах.

Таблица 2.4. Тригонометрические и гиперболические функции

Функция	Синтаксис Maple	Функция	Синтаксис Maple
$\sin(x)$	`sin(x)`	$\mathrm{sh}(x)$	`sinh(x)`
$\cos(x)$	`cos(x)`	$\mathrm{ch}(x)$	`cosh(x)`
$\mathrm{tg}(x)$	`tan(x)`	$\mathrm{th}(x)$	`tanh(x)`
$\sec(x)$	`sec(x)`	$\mathrm{sech}(x)$	`sech(x)`
$\mathrm{cosec}(x)$	`csc(x)`	$\mathrm{cosech}(x)$	`csch(x)`
$\mathrm{ctg}(x)$	`cot(x)`	$\mathrm{cth}(x)$	`coth(x)`

Задание обратных тригонометрических и обратных гиперболических функций представлено табл. 2.5.

Таблица 2.5. Обратные тригонометрические и гиперболические функции

Функция	Синтаксис Maple	Функция	Синтаксис Maple
$\arcsin(x)$	`arcsin(x)`	$\mathrm{arcsh}(x)$	`arcsinh(x)`
$\arccos(x)$	`arccos(x)`	$\mathrm{arcch}(x)$	`arccosh(x)`
$\mathrm{arctg}(x)$	`arctan`	$\mathrm{arcth}(x)$	`arctanh(x)`
$\mathrm{arcsec}(x)$	`arcsec(x)`	$\mathrm{arcsech}(x)$	`arcsech(x)`
$\mathrm{arccosec}(x)$	`arccsc(x)`	$\mathrm{arccosech}(x)$	`arccsch(x)`
$\mathrm{arcctg}(x)$	`arccot(x)`	$\mathrm{arccth}(x)$	`arccoth(x)`

Задание в Maple функций Бесселя, эллиптических интегралов, дельта-функции Дирака, функции Хевисайда и других специальных функций можно найти в справочной системе.

Замечание

Справку обо всех имеющихся в Maple функциях можно получить, выполнив команду `?inifunction`.

2.2. Команды преобразования выражений

Технология работы в Maple заключается в том, что пользователь создает переменные, присваивает им символьные выражения и производит над ними некоторые действия в соответствии с алгоритмом решения поставленной задачи, использую стандартные функции или написанные собственные процедуры.

Синтаксис вызова стандартной команды следующий:

```
команда(пар_1, пар_2, ...,пар_n);
```

или

```
команда(пар_1, пар_2, ...,пар_n):
```

Здесь `команда` — это имя вызываемой функции, а `пар_1`, `пар_2`, `...` означают необходимые для выполнения команды параметры, которые могут быть переменными или даже выражениями, причем их тип должен соответствовать типу параметров используемой функции. Отметим, что первая форма задания команды (с завершающей точкой с запятой) осуществляет отображение результатов ее выполнения в области вывода, тогда как при второй форме (с завершающим двоеточием) команда выполняется, но никакого вывода результатов не происходит.

Система обозначений функций в Maple интуитивно проста, поэтому обыкновенно имя функции соответствует действию, которое она выполняет (следует учесть, что все имена заданы на английском языке). Например, ясно, что функция с именем `simplify()` осуществляет некоторые упрощения над выражением, заданным в качестве ее параметра.

Для некоторых команд существуют две формы: *активная* и *пассивная*. В случае вызова активной формы команды, которая немедленно будет выполнена, ее имя начинается со строчной буквы. Пассивная форма команды не выполняется немедленно ядром Maple, а просто в области вывода отображает математическую запись того, что она может сделать. Ее имя начинается с прописной буквы. В дальнейшем, если в операторе присваивания для некоторой переменной в правой части задана пассивная форма команды, то командой `value()` ее можно вычислить. Однако основное предназначение пассивных форм команд — использование их как средства документирования производимых действий в обычной математической нотации. Примерами команд с двумя формами являются команда дифференцирования (`diff` и `Diff`), интегрирования (`int` и `Int`) и др.

Пример 2.10. Пассивная и активная формы команд

```
> g:=Int(sin(x)^2,x);
```

$$g := \int \sin(x)^2\, dx$$

```
> g=int(sin(x)^2,x);
```

$$\int \sin(x)^2\, dx = -\frac{1}{2}\cos(x)\sin(x) + \frac{1}{2}x$$

```
> value(g);
```

$$-\frac{1}{2}\cos(x)\sin(x) + \frac{1}{2}x$$

Замечание

Командой `?inert` можно отобразить страницу Справки, содержащей информацию об активной и пассивной форме некоторых функций Maple.

Команды и функции, являющиеся частью ядра системы Maple, всегда доступны пользователю, тогда как для вызова других команд и функций необходимо подключить библиотеку или пакет, в которых они расположены. Для этого используются команды `readlib()` и `with()`. Первая подключает библиотеку, вторая — пакет. Параметром этих команд является имя библиотеки или пакета, функции которых пользователь желает использовать.

Наиболее часто используемые при аналитических преобразованиях команды и функции Maple располагаются в его системном ядре — части программного обеспечения системы аналитических вычислений, постоянно находящейся в памяти компьютера. К таким командам относятся команды, выполняющие разнообразные преобразования выражений, получающие решение уравнений и систем уравнений, дифференцирующие функции и т. д. В данном разделе вводятся команды, наиболее часто используемые при выполнении аналитических вычислений.

2.2.1. Упрощение выражения: *simplify()*

Начнем с команды упрощения выражений — команды `simplify()`. Эта команда предназначена для упрощения разнообразных выражений, включающих рациональные дроби (алгебраические выражения), содержащих тригонометрические, обратные тригонометрические функции, логарифмы и экспоненты, т. е. с ее помощью можно попытаться упростить выражение, составленное из элементарных функций. Почему попытаться? Просто потому, что Maple может его упростить, а может и не упростить, так как он использует свои внутренние алгоритмы упрощения, результат выполнения которых может не совсем соответствовать взглядам пользователя на то, как он хотел бы упростить выражение и в каком виде его получить. Вообще, задача упрощения во всех системах аналитических вычислений — это достаточно сложная проблема. В одном контексте вычислений какое-то преобразование считается упрощением, а в другом то же самое преобразование может и не считаться упрощением. Например, при решении тригонометрических уравнений не всегда рационально заменять $sin(x)^2 + cos(x)^2$ на 1, хотя это явное упроще-

ние. Иногда надо сделать наоборот: единицу представить в виде суммы квадратов синуса и косинуса, и вот тогда такое "упрощение" приведет к упрощению всего уравнения, позволит разложить его на множители и решить поставленную задачу.

Эта команда имеет несколько форм вызова, отличающихся наличием параметров, управляющих процедурой упрощения. Ее самый простой синтаксис имеет следующий вид:

```
simplify(выражение);
```

В скобках в качестве параметра передается выражение, подлежащее упрощению. Команда `simplify()` ищет в выражении вызовы функций, квадратные корни, радикалы и степени и инициализирует подходящие процедуры упрощения. Реально команда `simplify()` реализована в виде набора процедур упрощения, хранящихся в основной библиотеке Maple. Мы перечислим часть из них, остальные можно найти в справке по этой команде (например, установив курсор в рабочем листе на ее имя и нажав клавишу <F1>): `` `simplify/exp` `` — для упрощения выражений с экспоненциальными функциями, `` `simplify/ln` `` — для упрощения выражений с логарифмами, `` `simplify/sqrt` `` — для упрощения выражений, содержащих квадратные корни, `` `simplify/trig` `` — для упрощения выражений с тригонометрическими функциями, `` `simplify/radical` `` — для упрощения выражений с радикалами (дробные степени), `` `simplify/power` `` — для упрощения выражений со степенями, экспонентами и логарифмами и т. д. По умолчанию Maple пытается использовать максимальный набор функций упрощения, подходящий к конкретному выражению.

В вызове команды можно задать конкретные процедуры упрощения, и тогда только они будут использоваться для упрощения заданного выражения, а не весь возможный, установленный по умолчанию набор. Такой вызов обеспечивается следующим синтаксисом команды:

```
simplify(выражение, n1, n2, ...);
```

Здесь `n1`, `n2` и т. д. являются именами процедур упрощения: `Ei`, `GAMMA`, `RootOf`, `@`, `hypergeom`, `ln`, `polar`, `power`, `radical`, `sqrt`, `trig`. Полную информацию о формулах упрощения при использовании перечисленных значений параметров можно получить с помощью команды `?simplify[имя]`, где `имя` — одно из значений параметров функции упрощения.

При упрощении выражения можно предположить, что все переменные в нем являются, например, положительными, или принадлежат некоторому отрезку действительных чисел. Это осуществляется заданием ключевого параметра `assume=свойство`. Форма вызова команды в этом случае имеет вид:

```
simplify(выражение, assume=свойство);
```

где параметр `свойство` может принимать одно из следующих значений: `complex` — комплексная область, `real` — действительная область, `positive` —

положительные действительные числа, `integer` — целые числа, `RealRange(a,b)` — интервал `(a,b)` действительных чисел.

Ниже представлены примеры использования команды упрощения выражений `simplify()`:

Пример 2.11. Упрощение выражений

```
> f:= ln(exp(x));
```

$$f := \ln(e^x)$$

```
> simplify(f);
```

$$\ln(e^x)$$

```
> simplify(f, ln, assume=real);
```

$$x$$

```
> a:=1/sqrt(5)*(((1+sqrt(5))/2)^3-((1-sqrt(5))/2)^3);
```

$$a := \frac{1}{5}\sqrt{5}\left(\left(\frac{1}{2}+\frac{1}{2}\sqrt{5}\right)^3 - \left(\frac{1}{2}-\frac{1}{2}\sqrt{5}\right)^3\right)$$

```
> simplify(a);
```

$$2$$

Обратим внимание на упрощение выражения `f`. Использование команды без параметров не упростило выражения $\ln(e^x)$, тогда как второй оператор с предположением о действительной области изменения переменной x упростил заданное выражение. При упрощении Maple предполагает, что там, где это возможно, переменные изменяются в области комплексных чисел. При таком предположении упростить выражение `f` действительно невозможно.

При вызове команды упрощения можно последним, или единственным, не считая самого упрощаемого выражения, параметром задать параметр с именем `symbolic`. В этом случае, если выражение содержит многозначные функции, например квадратный корень, то относительно таких функций будет осуществлено формальное символическое упрощение. Это означает, что не будет приниматься во внимание различное поведение многозначных функций при нахождении их аргумента в разных областях комплексной плоскости или действительной оси. Так, в случае с функцией $y=\sqrt{x^2}$ при упрощении следует учитывать, положительна или отрицательна неизвестная x; задание параметра `symbolic` снимает эту неопределенность, используя формальное правило: квадратный корень из квадрата какой-либо величины равен этой величине.

Пример 2.12. Упрощение с предположением

```
> f:=(sqrt(x^2));
```

$$f := \sqrt{x^2}$$

```
> simplify(f);
```

$$\mathrm{csgn}(x)\,x$$

```
> simplify(f,assume=real);
```

$$|x|$$

```
> simplify(f,assume=positive);
```

$$x$$

```
> simplify(f,symbolic);
```

$$x$$

Замечание

Следует с осторожностью использовать параметр `symbolic` в функции `simplify()`, так как в большинстве случаев результат упрощения не верен на всей комплексной плоскости, а также не всегда известно, какая ветвь многозначной функции использовалась при упрощении.

Команда упрощения позволяет задать правила упрощения в виде равенств. Эти правила задаются вторым параметром, который должен иметь следующий вид:

```
{равенство1, равенство2, ...}
```

Если какое-то выражение при упрощении должно равняться нулю, то такое правило можно задать, просто внеся выражение без знака равенства в список правил:

```
> g:=a^2+b^2+3*c;
```

$$g := a^2 + b^2 + 3\,c$$

```
> simplify(g,{b^2,a^2+c=1});
```

$$2\,c + 1$$

В этом демонстрационном примере предполагается, что квадрат величины b равен 0.

Замечание

Забегая немного вперед, скажем, что конструкция в фигурных скобках с заданным через запятую списком выражений определяет объект Maple, который представляет собой множество в его математическом смысле.

Использование собственных правил для упрощения тригонометрических выражений позволяет получить именно тот его вид, который необходим для дальнейшей работы, так как третьим параметром можно определить, в какой последовательности должны отображаться неизвестные в упрощенном выражении. Этот параметр задается в двух формах: в виде множества и в виде списка. (О множестве мы упомянули при определении правил упрощения пользователя, а список — это тоже объект Maple, который пока можно считать как список выражений через запятую, заключенный в квадратные скобки.) Так вот, если он задан в виде множества, то алгоритм упрощения

сортирует в выражении неизвестные по убыванию их степени в слагаемых выражения, учитывая степени всех неизвестных, а потом начинает упрощения в соответствии с заданными правилами. В случае со списком — сначала выражение сортируется по степеням первой неизвестной в списке, затем упрощается в соответствии с заданными правилами, затем полученное выражение сортируется по степеням второй неизвестной списка и упрощается и т. д.

Пример 2.13. Упрощение в соответствии с правилами пользователя

```
> equ := {sin(x)^2 + cos(x)^2 = 1};
  e := sin(x)^3 - 11*sin(x)^2*cos(x) + 3*cos(x)^3 - sin(x)*cos(x) + 2;
```

$$equ := \{ \sin(x)^2 + \cos(x)^2 = 1 \}$$

$$e := \sin(x)^3 - 11 \sin(x)^2 \cos(x) + 3 \cos(x)^3 - \sin(x) \cos(x) + 2$$

```
> simplify(e, equ, [sin(x),cos(x)]);
```

$$14 \cos(x)^3 - \sin(x) \cos(x) + 2 - \sin(x) \cos(x)^2 + \sin(x) - 11 \cos(x)$$

```
> simplify(e, equ, [cos(x),sin(x)]);
```

$$\sin(x)^3 - 14 \sin(x)^2 \cos(x) - \sin(x) \cos(x) + 2 + 3 \cos(x)$$

```
> simplify(e, equ, {sin(x),cos(x)});
```

$$\sin(x)^3 - 14 \sin(x)^2 \cos(x) - \sin(x) \cos(x) + 2 + 3 \cos(x)$$

```
> simplify(e, equ, {cos(x),sin(x)});
```

$$\sin(x)^3 - 14 \sin(x)^2 \cos(x) - \sin(x) \cos(x) + 2 + 3 \cos(x)$$

Замечание

Подробнее познакомиться с использованием собственных правил упрощения можно на странице Справки, отображаемой командой `?simplify[siderels]`.

2.2.2. Раскрытие скобок в выражении: *expand()*

Основное назначение команды `expand()` — представить произведение в виде суммы, т. е. данная команда раскрывает скобки в алгебраическом выражении. Она выполняется для любого полинома. Для частного двух полиномов (рациональная алгебраическая дробь) эта команда раскрывает скобки в числителе и делит каждый член полученного выражения на знаменатель, с которым она не производит никаких преобразований.

Кроме того, данная команда умеет работать с большинством математических функций и знает, как раскрывать скобки в выражениях, содержащих следующие функции: sin(x), cos(x), tg(x), sh(x), ch(x), th(x), ln(x), exp(x), abs(x), специальные математические функции и др.

Эта команда имеет следующий синтаксис:

```
expand(выр, выр1, выр2, ..., вырn);
```

где выр является выражением, в котором необходимо раскрыть скобки, а необязательные параметры выр1, выр2, ..., вырn указывают системе, что в данных выражениях в заданном преобразуемом выражении выр раскрывать скобки не надо.

Пример 2.14. Представление произведений в виде суммы

```
> expand((x+1)*(x+2));
```

$$x^2+3\,x+2$$

```
> expand((x+1)^3/(x+2)^2);
```

$$\frac{x^3}{(x+2)^2}+\frac{3\,x^2}{(x+2)^2}+\frac{3\,x}{(x+2)^2}+\frac{1}{(x+2)^2}$$

```
> expand(sin(x+y));
```

$$\sin(x)\cos(y)+\cos(x)\sin(y)$$

```
> expand(exp(a+ln(b)));
```

$$e^a\,b$$

```
> expand((x+1)^2*(y+z), x+1);
```

$$(x+1)^2\,y+(x+1)^2\,z$$

Как видно из этого примера, данная функция знает правила преобразования тригонометрических выражений, выражений с экспоненциальными функциями, полиномами и другими функциями.

Совет

Может показаться, что команда simplify() одна из самых полезных команд Maple. Однако, это не совсем так. Как мы видели, эта команда упрощает выражение в соответствии со своими внутренними представлениями о том, что считать более простым видом выражения. Например, она всегда считает, что сумма проще произведения, хотя в некоторых случаях может оказаться как раз наоборот. Команду expand(), алгоритм которой достаточно гибок, можно также использовать для упрощения вида выражения, например, если необходимо оставить не раскрытым какое-то подвыражение.

2.2.3. Разложение полинома на множители: *factor()*

Основное предназначение команды factor() — разложить на множители полином от нескольких переменных. Под *полиномом* в Maple понимается выражение, содержащее неизвестные величины, в котором каждый член представлен в виде произведения целых неотрицательных степеней неизвестных величин с числовым или алгебраическим коэффициентом, т. е. коэффициент может быть целым, дробным, с плавающей точкой, комплекс-

ным числом и даже представлять собой алгебраическое выражение с другими переменными:

```
> factor(x^3*y-x^3*b-x^2*a*y+x^2*a*b+2*x^2*y^2-2*x^2*y*b-2*x*y^2*a
        +2*x*y*a*b+y^3*x-y^2*x*b-y^3*a+y^2*a*b);
```

$$(x+y)^2(x-a)(y-b)$$

Замечание

Неизвестная в полиноме может быть представлена обращением к математической функции, параметр которой есть неизвестная:

```
> factor(cos(y)^2-2*sin(x)*cos(y)+sin(x)^2);
```

$$(\sin(x)-\cos(y))^2$$

Относительно этой команды следует помнить одно правило: она раскладывает полином на множители над числовым полем, которому принадлежат коэффициенты полинома. Это означает, если все коэффициенты целые, то и в получаемых сомножителях будут только целые коэффициенты и не обязательно будут получены линейные сомножители. Второй необязательный параметр этой команды служит для указания, над каким числовым полем следует осуществлять разложение полинома. Он может иметь значение `real`, `complex`, а также один радикал или список/множество радикалов. Пример 2.15 демонстрирует результаты разложения одного и того же полинома над разными полями.

Пример 2.15. Разложение полинома над разными полями

```
> factor(x^3+2);               # над полем целых чисел
                               # (целые коэффициенты)
```

$$x^3+2$$

```
> factor(x^3+2.0);             # над полем вещественных чисел
                               # (вещественный коэффициент)
```

$$(x+1.259921050)(x^2-1.259921050\,x+1.587401052)$$

```
> factor(x^3+2,real);          # над полем вещественных чисел
                               # (параметр real)
```

$$(x+1.259921050)(x^2-1.259921050\,x+1.587401052)$$

```
> factor(x^3+2,complex);       # над полем комплексных чисел
                               # (параметр complex)
```

$$(x+1.259921050)(x-.6299605249+1.091123636\,I)(x-.6299605249-1.091123636\,I)$$

```
> factor(x^3+2,2^(1/3));       # над полем целых и радикала 2^(1/3)
                               # (параметр определяет поле с радикалом)
```

$$(x^2-2^{(1/3)}x+2^{(2/3)})(x+2^{(1/3)})$$

При применении этой же команды к алгебраической рациональной дроби (отношение двух полиномов) сначала осуществляется приведение дроби к нормальной форме (сокращение общих множителей числителя и знаменателя), а после этого и числитель, и знаменатель раскладываются на множители (с учетом поля коэффициентов):

```
> rational_exp:=(x^15-y^15)/(x^8-y^8);
```

$$rational_exp := \frac{x^{15}-y^{15}}{x^8-y^8}$$

```
> factor(rational_exp);
```

$$\frac{(y^2+x\,y+x^2)\,(y^4+y^3\,x+y^2\,x^2+y\,x^3+x^4)\,(y^8-y^7\,x+y^5\,x^3-y^4\,x^4+y^3\,x^5-y\,x^7+x^8)}{(y+x)\,(x^2+y^2)\,(x^4+y^4)}$$

2.2.4. Сокращение алгебраической дроби: *normal()*

Назначение команды `normal()` — привести выражение, содержащее алгебраические дроби, к общему знаменателю и упростить полученную алгебраическую дробь, сократив и числитель, и знаменатель на наибольший общий делитель. Она имеет две формы вызова:

```
normal(f);
normal(f, expanded);
```

где `f` — алгебраическая дробь, а параметр `expanded` служит для указания того, что после сокращения дроби в числителе и знаменателе раскрываются скобки.

Пример 2.16. Сокращение алгебраических дробей

```
> fr:=1/(x+1)+1/x+x/(x+1);
```

$$fr := \frac{1}{x+1}+\frac{1}{x}+\frac{x}{x+1}$$

```
> fr:=normal(fr);
```

$$fr := \frac{x+1}{x}$$

```
> f:=(x^2-y^2)/(x-y)^3;
```

$$f := \frac{x^2-y^2}{(-y+x)^3}$$

```
> normal(f);
```

$$\frac{y+x}{(-y+x)^2}$$

```
> normal(2/x + y/3 = 0);
```

$$\frac{1}{3}\,\frac{6+x\,y}{x}=0$$

Если параметр `f` задан в виде списка, множества, последовательности, ряда, уравнения, отношения или функции, то команда `normal()` последовательно применяется к компонентам `f`. Например, для уравнения это означает, что процедура сокращения применяется и к правой, и к левой части уравнения. В случае ряда, это означает, что упрощаются коэффициенты ряда, а в случае выражения с несколькими функциями, аргументы которых представлены алгебраическими дробями, процедура сокращения применяется к аргументу каждой функции:

```
> s:=sin(x/(x+1)-x)^2+cos(-x/(x+1)+x);
```

$$s := \sin\left(\frac{x}{x+1} - x\right)^2 + \cos\left(-\frac{x}{x+1} + x\right)$$

```
> normal(s);
```

$$\sin\left(\frac{x^2}{x+1}\right)^2 + \cos\left(\frac{x^2}{x+1}\right)$$

```
> normal(2/x + y/3 = x/y+y/x+2);
```

$$\frac{1}{3}\frac{6+x\,y}{x} = \frac{x^2+y^2+2\,x\,y}{x\,y}$$

Обратим внимание на то, что вторая команда `normal()` в приведенных примерах применяется к уравнению (`equation`) — типу Maple, обсуждение которого отложим до знакомства с командой решения уравнений и неравенств.

2.2.5. Приведение нескольких членов выражения к одному: *combine()*

Команда `combine()` приводит несколько членов в выражении, представленном суммой, произведением или степенями неизвестных, к одному члену, используя разнообразные правила, которые, по существу, противоположны правилам, применяемым командой `expand()`. Например, рассмотрим известное тригонометрическое соотношение:

$$\sin(a + b) = \sin(a)\cos(b) + \cos(a)\sin(b)$$

Команда `expand()` использует его слева направо, тогда как команда `combine()` действует наоборот, представляя сумму произведений синусов и косинусов в виде одной тригонометрической функции, но с аргументом, являющимся комбинацией аргументов тригонометрических функций в преобразуемом выражении:

```
> g:=sin(a+b)^2;
```

$$g := \sin(a + b)^2$$

```
> s:=expand(g);
```

$$s := \sin(a)^2\cos(b)^2 + 2\sin(a)\cos(b)\cos(a)\sin(b) + \cos(a)^2\sin(b)^2$$

```
> combine(s);
```

$$\frac{1}{2}-\frac{1}{2}\cos(2\,a+2\,b)$$

Обратите внимание, что команда `combine()` преобразовала выражение `s` не к исходному выражению `g`, которое мы раскрыли функцией `expand()`. Это связано с тем, что Maple осуществляет приведение членов выражения по своим внутренним алгоритмам, которые завершаются, как только получилось (или не получилось) представление в соответствии с идеологией команды `combine()`. В нашем примере — представление через тригонометрическую функцию с аргументом, являющимся линейной комбинацией аргументов тригонометрических функций преобразуемого выражения. Если, однако, желательно получить исходный вид выражения `g`, то следует воспользоваться командой подстановки `subs()`, параметры которой определяют, что на что следует заменить в выражении:

```
> subs(cos(2*a+2*b)=-2*sin(a+b)^2+1,f);
```

$$\sin(a+b)^2$$

(Команда `subs()` подробно описывается в разделе 2.3.)

Команда `combine()` "знает" практически все правила преобразования элементарных математических функций. Если вторым ее параметром задать одно из следующих имен:

abs	exp	piecewise	Psi	Signum
arctan	icombine	polylog	radical	trig
conjugate	ln	power	range	

которые соответствуют используемым в Maple функциям, то при преобразовании выражения будут применяться только правила преобразования соответствующих функций. Для функций, правила преобразования которых зависят от значения их аргументов (`arctan`) или которые имеют ограничения на значения аргументов (`ln`, `radical`), можно задать третий параметр `symbolic`, который будет предписывать функции `combine()` не обращать внимания на интервалы изменения аргументов подобных функций, а осуществлять формальные символические преобразования в соответствии с формулами преобразования этих функций. Преобразования с большинством функций в основном ясны. Например, если используется опция `ln`, то к выражению применяются следующие преобразования, известные из школьного курса математики:

```
a*ln(x)        ==>    ln(x^a)  (если a*argument(x)=argument(x^a))
ln(x)+ln(y)    ==>    ln(x*y)  (если argument(x*y) =
                                  argument(x) + argument(y))
```

Особо отметим преобразование с опцией icombine, которая пытается представить произведение степеней целых чисел таким образом, чтобы в полученном произведении степени не имели общих множителей:

```
> combine(4^a * 6^b * 12^c * 5^d, power);
```

$$4^a\,6^b\,12^c\,5^d$$

```
> combine(4^a * 6^b * 12^c * 5^d, icombine);
```

$$2^{(2\,a+b+2\,c)}\,3^{(c+b)}\,5^d$$

Замечание

Более подробную информацию об опциях команды combine() можно получить, выполнив команду ?combine[опция].

2.2.6. Приведение подобных членов: *collect()*

Команда collect() работает с обобщенными полиномами нескольких переменных — полиномами, в которых в качестве неизвестных могут выступать функции с аргументами, являющимися неизвестными величинами Maple. Синтаксис этой команды имеет три формы:

```
collect(выражение, x)
collect(выражение, x, form, func)
collect(выражение, x, func)
```

В них параметр x представляет имя неизвестной величины, список или множество неизвестных в случае полинома нескольких переменных или имя функции с аргументом-неизвестной в выражении, представленном первым параметром выражение, и относительно степеней которой осуществляется приведение коэффициентов.

Замечание

Команда collect() различает не только целые, но и положительные и отрицательные дробные степени неизвестной, т. е. при всех степенях будут отдельно приведены подобные члены.

Пример 2.17. Приведение коэффициентов в выражении

```
> g := int(x^2*(exp(x)+sin(x)),x);
```

$$g := x^2\,e^x - 2\,x\,e^x + 2\,e^x - x^2\cos(x) + 2\cos(x) + 2\,x\sin(x)$$

```
> collect(g,x);
```

$$(-\cos(x) + e^x)\,x^2 + (-2\,e^x + 2\sin(x))\,x + 2\cos(x) + 2\,e^x$$

```
> collect(g,exp(x));
```

$$(2 + x^2 - 2\,x)\,e^x + 2\cos(x) + 2\,x\sin(x) - x^2\cos(x)$$

```
> collect(g,cos(x));
```

$$(-x^2+2)\cos(x)+x^2\,e^x-2\,x\,e^x+2\,e^x+2\,x\sin(x)$$

Команды `collect()` примера 2.17 для одного и того же выражения осуществляют приведение коэффициентов относительно разных его неизвестных компонентов.

Параметр `form` имеет смысл для полиномов от нескольких переменных и определяет алгоритм приведения подобных членов, причем неизвестные, при степенях которых приводятся подобные члены, должны быть заданы в виде списка или множества. Он имеет два значения: `recursive` (значение по умолчанию) и `distributed`. Первый инициирует следующий алгоритм: приводятся подобные члены при степенях первой неизвестной в списке, далее в полученных коэффициентах приводятся подобные члены относительно степеней второй неизвестной в списке и т. д. Если при этом значении параметра `form` неизвестные полинома, относительно которых приводятся подобные члены, заданы в виде множества, то порядок приведения определяется системой Maple и может меняться от сеанса к сеансу. Значение `distributed` указывает на приведение коэффициентов при членах, содержащих всевозможные произведения степеней неизвестных в списке или множестве, причем суммарная степень всех переменных возрастает от наименьшей к наибольшей.

Пример 2.18. Алгоритмы приведения для полиномов нескольких переменных

```
> p := x*y-a^2*x*y+y*x^2-a*y*x^2+x+a*x; # полином двух переменных
```

$$p:=x\,y-a^2\,x\,y+y\,x^2-a\,y\,x^2+x+a\,x$$

```
> collect( p, [x,y], recursive);
```

$$(1-a)\,y\,x^2+((1-a^2)\,y+1+a)\,x$$

```
> collect( p, [y,x], recursive);
```

$$((1-a)\,x^2+(1-a^2)\,x)\,y+(1+a)\,x$$

```
> collect( p, {x,y}, recursive);
```

$$(1-a)\,y\,x^2+((1-a^2)\,y+1+a)\,x$$

```
> collect( p, {x,y}, distributed);
```

$$(1+a)\,x+(1-a^2)\,x\,y+(1-a)\,y\,x^2$$

```
> collect( p, [x,y], distributed);
```

$$(1+a)\,x+(1-a^2)\,x\,y+(1-a)\,y\,x^2$$

Параметр `func` определяет имя команды, которая применяется к полученным в результате коэффициентам при соответствующих степенях неизвестных. Обычно используют команды `simplify()` и `factor()`.

Пример 2.19. Задание функции, применяемой к полученным коэффициентам

```
> f := a^3*x-x+a^3+a;
```

$$f := a^3\,x - x + a^3 + a$$

```
> collect(f,x);
```

$$(a^3 - 1)\,x + a^3 + a$$

```
> collect(f,x,factor); # разложение на множители коэффициентов при x
```

$$(a-1)\,(a^2+a+1)\,x + a\,(a^2+1)$$

```
> collect(p,[x,y],distributed,factor); # полином двух переменных
```

$$(1+a)\,x - (a-1)\,(1+a)\,x\,y + (1-a)\,y\,x^2$$

```
> collect(p,[x,y],recursive,factor);
```

$$(1-a)\,y\,x^2 + (-(a-1)\,(1+a)\,y + 1 + a)\,x$$

Замечание

Алгоритмы приведения подобных членов команды `collect()` никоим образом не сортируют коэффициенты в полученном полиноме. Чтобы их упорядочить, следует обратиться к команде `sort()`, описание которой можно найти далее в этой же главе.

2.2.7. Рационализация дробей: *rationalize()*

Под рационализацией дробей понимается избавление от иррациональности в знаменателе. Команда `rationalize()` и производит именно такое преобразование над числовыми и алгебраическими дробями. Причем в последнем случае принимается во внимание только знаменатель в виде полинома. Эта команда может рационализировать алгебраическую дробь, знаменатель которой содержит трансцендентные функции типа `sin()`, `exp()`, `ln()` и т. п. Однако если их аргумент является дробью с иррациональностями в знаменателе, то эти конструкции не участвуют в процессе рационализации.

Пример 2.20. Рационализация дробных выражений

```
> ex1:=2*(1+2^(1/3))/(2-sqrt(2));
```

$$ex1 := 2\,\frac{1+2^{(1/3)}}{2-\sqrt{2}}$$

```
> rationalize(ex1);
```

$$(1+2^{(1/3)})\,(2+\sqrt{2})$$

```
> [ x/(x+sqrt(1+sqrt(3))),(x+y)/(x*y+sqrt(3)+sqrt(7))];
```

$$\left[\frac{x}{x+\sqrt{1+\sqrt{3}}}, \frac{x+y}{x\,y+\sqrt{3}+\sqrt{7}}\right]$$

```
> rationalize(%);
```

$$\left[\frac{x\,(x-\sqrt{1+\sqrt{3}}\,)\,(x^2-1+\sqrt{3}\,)}{x^4-2\,x^2-2},\frac{(x+y)\,(x\,y+\sqrt{3}-\sqrt{7}\,)\,(x^2\,y^2-4-2\,x\,y\,\sqrt{3}\,)}{x^4\,y^4-20\,x^2\,y^2+16}\right]$$

```
> [(x+y)/(x+sqrt(y)), x*y/(x+sqrt(x+sqrt(3)))];
```

$$\left[\frac{x+y}{x+\sqrt{y}},\frac{x\,y}{x+\sqrt{x+\sqrt{3}}}\right]$$

```
> rationalize(%);
```

$$\left[\frac{(x+y)\,(x-\sqrt{y}\,)}{x^2-y},\frac{x\,y\,(x-\sqrt{x+\sqrt{3}}\,)\,(x^2-x+\sqrt{3}\,)}{x^4-2\,x^3+x^2-3}\right]$$

```
> 1/(1+root(sin(1/(1-sqrt(eta))),3));
```

$$\frac{1}{1+\sin\left(\frac{1}{1-\sqrt{\eta}}\right)^{(1/3)}}$$

```
> rationalize( %);
```

$$\frac{1-\sin\left(\frac{1}{1-\sqrt{\eta}}\right)^{(1/3)}+\sin\left(\frac{1}{1-\sqrt{\eta}}\right)^{(2/3)}}{1+\sin\left(\frac{1}{1-\sqrt{\eta}}\right)}$$

Замечание

Обратите внимание, что вторая команда `rationalize()` применяется к списку выражений. Практически все команды и функции Maple могут применяться к списку. В этом случае их действие распространяется на каждый элемент списка.

2.2.8. Ограничения на неизвестные: *assume()*

Осуществляя разнообразные математические выводы (упрощение выражений, доказательство теорем) в обычной манере на листе бумаги, зачастую приходится делать те или иные предположения относительно некоторых величин, которые фигурируют в наших исследованиях. Одни ограничения логически вытекают из области определения независимых переменных, входящих в выражения, другие мы накладываем сами. В любом случае, зачастую без введения определенных ограничений на некоторые выражения ничего нельзя сказать о свойствах математических объектов, в которых они фигурируют. Система Maple, стремясь быть помощником математика, предлагает широкий спектр команд для введения и проверки ограничений, наложенных на некоторые неизвестные или даже целые выражения. Введенные ограничения используются командами и функциями Maple, например `simplify()`, `sqrt()`, для получения более простого ответа, если введенные ограничения позволяют это.

Команда `assume()` накладывает ограничения на неизвестные величины Maple. Она имеет следующий синтаксис:

```
assume(x, свойство);
```

Здесь `x` представляет любую неопределенную переменную Maple или выражение с такими переменными, а параметр `свойство` может принимать значения, равные названиям свойств (специальным символьным именам, зарезервированным системой Maple для задания разнообразных ограничений на переменную или выражение, определенные первым параметром), имени типа данных и числовому диапазону. Некоторые из наиболее употребительных свойств перечислены в табл. 2.6.

Таблица 2.6. *Свойства числовых переменных и выражений*

Название свойства	Описание
`negative`	Отрицательные вещественные числа из интервала (-∞,0) (нуль не включается)
`nonnegative`	Неотрицательные вещественные числа из интервала [0,∞) (нуль включается)
`positive`	Положительные вещественные числа из интервала (0,∞) (нуль не включается)
`natural`	Натуральные числа (целые, большие или равные 0)
`posint`	Целые строго большие 0
`odd`	Нечетные числа
`even`	Четные числа
`complex`	Комплексные числа
`NumeralNonZero`	Комплексные числа, исключая 0
`real`	Вещественные числа
`rational`	Рациональные числа (дроби и целые)
`irrational`	Иррациональные числа
`integer`	Целые числа
`fraction`	Только дробные числа
`prime`	Простые числа

Замечание

Существуют свойства для задания ограничений на функции и матрицы. Познакомиться со всем перечнем используемых в Maple свойств и их имен можно на странице Справки, которая отображается командой `?property`.

Пару параметров (`x, свойство`) можно заменить математическим отношением, если, конечно, это возможно. Например, (`x,negative`) соответствует отношению `x<0`, (`x,nonnegative`) соответствует `x>=0` и т. д.

Если на переменную наложены ограничения, то в результатах выполнения действий над выражениями, в которые входит эта переменная, сразу же за ее именем по умолчанию отображается символ тильда (~). Эту функциональность по умолчанию можно изменить на следующие:

- либо вообще не информировать пользователя, что на переменную наложены ограничения, и она будет продолжать отображаться как и все переменные без ограничений (команда **Options** ➤ **Assumed Variables** ➤ **No Annotation**);
- либо в области вывода, если отображаются результаты, в которых присутствует переменная с наложенными ограничениями, словесно сообщается, на какие переменные наложены ограничения (команда **Options** ➤ **Assumed Variables** ➤ **Phrase**).

Пример 2.21. Способы отображения переменных с ограничениями

```
> assume(a>0);
> ln(a^2); # Отображение по умолчанию
```

$$2\ln(a\sim)$$

```
> ln(a^2); # Режим не информировать пользователя
```

$$2\ln(a)$$

```
> ln(a^2); # Словесное сообщение
```

$$2\ln(a)$$

with assumptions on a

Замечание

Вернуться в режим отображения переменных с наложенными ограничениями по умолчанию можно командой **Options** ➤ **Assumed Variables** ➤ **Trailing Tildes**.

Команда `assume()` может получать несколько пар (`x,свойство`) или математических отношений в качестве своих параметров. В этом случае все заданные ограничения действуют одновременно. Поэтому наложение ограничений в виде

```
> assume(x>1,x<2);
```

соответствует тому, что переменная `x` может изменяться только в интервале `(1,2)`.

Новое ограничение, накладываемое новой командой `assume()` на переменную, отменяет все предыдущие ограничения. Поэтому последовательное задание ограничений двумя командами:

```
> assume(x>1);
> assume(x<2);
```

соответствует предположению, что значение переменной `x` не превосходит числа `2`, а не тому, что значение этой переменной должно лежать в интервале `(1,2)`.

Если по ходу решения задачи необходимо постепенно добавлять ограничения на переменную, то можно использовать команду `additionally()`, параметры которой полностью соответствуют параметрам команды `assume()`. В этом случае ограничения, определенные командой `additionally()`, добавляются к ограничениям, введенным командой `assume()` и предыдущими командами `additionally()`:

```
> assume(x>1); # В последующих вычислениях предполагается x>1
        (какие-то вычисления)
> additionally(x<=2); # Теперь предполагается, что 1<x<=2
```

Для снятия всех наложенных ранее на переменную предположений следует этой переменной просто присвоить ее же символьное имя (имя переменной, заключенное в одинарные кавычки). Чтобы снять все ограничения для переменной `x` предыдущих примеров, следует просто выполнить следующую операцию присваивания:

```
> x:='x';
```

Однако если переменная с наложенными ограничениями использовалась в выражениях, то простое присваивание имени переменной самой переменной не снимает ограничения на переменную в этих выражениях. Пример 2.22 иллюстрирует подобную ситуацию.

Пример 2.22. Снятие ограничений с переменной

```
> assume(a>0);
> g:=sqrt(a^2);
```

$$g := a\sim$$

```
> a:='a': a;
```

$$a$$

```
> g;
```

$$a\sim$$

Как видим, снятие всех наложенных на переменную `a` ограничений не снимает, однако, этих ограничений с переменной `a` в выражении `g`. Чтобы снять ограничения с этой переменной, следует до команды снятия ограничений с переменной воспользоваться командой подстановки `subs()` и первым параметром указать замену переменной `a` на ее символьное имя `'a'`.

Пример 2.23. Снятие ограничений с переменной в выражении

```
> assume(a>0);
> g:=sqrt(a^4);
```

$$g := a\sim^2$$

```
> g;
```

$$a\sim^2$$

```
> g:=subs(a='a',g);
```

$$g := a^2$$

```
> a:='a';
```

$$a := a$$

```
> g;
```

$$a^2$$

С помощью функции `is()` можно определить, удовлетворяет ли некоторая переменная рабочего листа определенному свойству. Эта функция возвращает значение `true`, если все возможные значения переменной соответствуют заданному свойству. Если хотя бы одно из возможных значений не соответствует заданному свойству, то функция `is()` возвращает `false`. Эта функция может возвратить значение `FAIL`, которое информирует пользователя, что невозможно определить, соответствует или нет заданная переменная заданному свойству. Это может произойти либо в результате недостаточности информации относительно ограничений на переменную, либо невозможности вычислить логические ограничения на переменную.

Пример 2.24. Удовлетворяет ли переменная заданным ограничениям

```
> assume(a>0);
> is(a>0);
```

true

```
> is(a<1);
```

false

```
> additionally(a<1);
> is(a<1);
```

true

Функция `coulditbe()` проверяет, может ли заданная переменная соответствовать заданному свойству. Она возвращает `true`, если хотя бы одно из возможных значений переменной может иметь заданное свойство, и `false` в противном случае. Смысл значения `FAIL` соответствует такому же значению для функции `is()`.

Пример 2.25. Может ли переменная удовлетворять заданным ограничениям

```
> assume(a>0);
> is(a>0);
```

$$true$$

```
> coulditbe(a=1);
```

$$true$$

```
> additionally(a<1);
> coulditbe(a=1);
```

$$false$$

Команда about() отображает информацию о наложенных ограничениях на неизвестную величину:

```
> about(a);
Originally a, renamed a~:
  is assumed to be: RealRange(Open(0),Open(1))
```

Как отмечалось в начале этого подраздела, многие функции и команды Maple используют информацию о наложенных на неизвестную величину ограничениях при выполнении символьных вычислений. Например, Maple не может вычислить следующий предел из-за неизвестности знака символьной переменной a:

```
> int(exp(a*x),x=0..infinity);
```

$$\lim_{x \to \infty} \frac{e^{(a\,x)} - 1}{a}$$

Стоит задать предположение о строгой положительности параметра a, и Maple тут же вычислит данный интеграл, который он свел к вычислению предела, зависящего от параметра:

```
> assume(a>0);
> int(exp(a*x),x=0..infinity);
```

$$\infty$$

2.3. Структура выражений и их вычисление

До настоящего момента мы достаточно подробно рассмотрели числовые и строковые типы данных (объекты) Maple и упомянули о двух других объектах: списке и множестве. Для полноценного понимания функционирования системы аналитических вычислений, включая хранимую структуру выражений, нам просто необходимо познакомиться с основными типами объектов, кроме чисел и строк, с которыми работает Maple.

2.3.1. Основные сложные типы данных

В отличие от чисел — простых объектов — объекты других типов, с которыми может работать Maple, представляют собой сложные типы данных, образуемые с помощью специальных синтаксических конструкций из выражений Maple (последовательность, список, множество) или обращением к специальной функции-конструктору (массив, таблица).

2.3.1.1. Последовательность выражений

Последовательность выражений, или просто *последовательность*, — это группа выражений Maple, разделенных запятыми.

Пример 2.26. Последовательности выражений

```
> s1:=1,2,sqrt(3),x;
```

$$s1 := 1, 2, \sqrt{3}, x$$

```
> s2:=x^2,ln(x)+4,x*y;
```

$$s2 := x^2, \ln(x) + 4, x\,y$$

```
> s3:=1,1,x,1,x,2;
```

$$s3 := 1, 1, x, 1, x, 2$$

```
> whattype(s2);
```

$$exprseq$$

Последовательность является самостоятельным объектом, не имеющим ничего общего с внешне похожими на него списками и множествами. Скажем так: этот объект является базовым объектом Maple, на основе которого строятся другие сложные объекты. Он обладает присущими ему свойствами: сохраняет порядок следования выражений. Это означает, если мы определили, например, последовательность `s1` с первым элементом, равным целой величине `1`, то в этой последовательности (если мы, конечно, ее не изменим) этот элемент всегда будет первым. Последовательность может содержать и повторяющиеся элементы, расположенные в произвольном порядке внутри последовательности (`s3` в примере 2.26). Последняя команда этого примера определяет тип объекта, заданного в качестве его параметра. Для последовательности ее тип имеет имя `exprseq` (сокращение от expression sequence).

Если этот тип данных передается в качестве параметра функции Maple, то каждый его элемент рассматривается как соответствующий параметр функции. Для многих операций Maple использование в качестве одного из операндов последовательности приводит к выполнению этой операции для каждого ее элемента в качестве соответствующего операнда и формирования результирующей последовательности. Использование последовательности переменных в левой части операции присваивания и последовательности

значений в правой приводит к множественному присваиванию с помощью одной операции. Все эти возможности иллюстрируются в примере 2.27.

Пример 2.27. Операции с последовательностями

```
> s1:=sin(x)*x^2,x;
```

$$s1 := \sin(x)\,x^2, x$$

```
> int(s1);
```

$$-x^2\cos(x) + 2\cos(x) + 2\,x\sin(x)$$

```
> s2:=1,2,3;
```

$$s2 := 1, 2, 3$$

```
> s3:=y||s2;
```

$$s3 := y1, y2, y3$$

```
> f,g,h:=s2;
```

$$f, g, h := 1, 2, 3$$

```
> f;
```

$$1$$

Последовательность можно мыслить, как последовательность выражений, перенумерованных натуральными числами, начиная с единицы: первый элемент имеет индекс 1, второй — 2 и т. д. В связи с этим, можно получить значение любого элемента последовательности, используя индексную форму записи — после имени переменной в квадратных скобках задать индекс элемента:

```
> s:=x,x^2,x^3;
```

$$s := x, x^2, x^3$$

```
> s[2];
```

$$x^2$$

Однако присвоить новое значение элементу последовательности с использованием индексной формы обращения нельзя:

```
> s[2]:=x^(-2);
Error, cannot assign to an expression sequence
Ошибка, нельзя присвоить значение элементу последовательности выражений
```

Для создания длинных последовательностей, элементы которых подчиняются некоей закономерности, можно использовать команду `seq()` и операцию повторения `$`. Команда `seq()` имеет две формы:

```
seq(f,i=m..n);
seq(f,i=x);
```

В них `f` — выражение, зависящее от переменной, имя которой определяется параметром `i`, `m` и `n` — числа, определяющие диапазон изменения перемен-

ной `i` с шагом 1, а `x` может быть списком, множеством, суммой, произведением или строкой. В последнем случае переменная `i` последовательно принимает значения, равные символам строки.

Пример 2.28. Формирование последовательностей командой seq()

```
> seq( sin(Pi*i/6), i=0..6);
```

$$0, \frac{1}{2}, \frac{1}{2}\sqrt{3}, 1, \frac{1}{2}\sqrt{3}, \frac{1}{2}, 0$$

```
> seq(x[k],k=3..5);
```

$$x_3, x_4, x_5$$

```
> seq(cat(k,"1"),k="string");
```

"s1", "t1", "r1", "i1", "n1", "g1"

Операция повторения `$` может быть унарной и бинарной. В первом случае она применяется к диапазону `m..n`, где `m` и `n` — числа, а во втором случае первым операндом является выражение, зависящее от переменной, которая указана во втором операнде, содержащем также операцию диапазон. Все случаи ее использования демонстрируются в примере 2.29.

Пример 2.29. Формирование последовательностей операцией $

```
> $ 2..5;
```

$$2, 3, 4, 5$$

```
> i^2 $ i = 2/3 .. 8/3;
```

$$\frac{4}{9}, \frac{25}{9}, \frac{64}{9}$$

```
> a[i] $ i = 1..3;
```

$$a_1, a_2, a_3$$

```
> x$4;
```

$$x, x, x, x$$

2.3.1.2. Списки и множества

Список — упорядоченная последовательность выражений, заключенная в квадратные скобки, а *множество* — неупорядоченная последовательность выражений, заключенная в фигурные скобки. Относительно множества следует отметить, что этот объект понимается в Maple точно так же, как и в математике. Если в последовательности присутствуют повторяющиеся элементы, то в множестве им будет соответствовать один элемент, тогда как в списке все повторяющиеся элементы существенны, т. е. список может содержать повторяющиеся элементы, стоящие на разных местах. При задании

множества порядок элементов не существен, главное, чтобы в последовательности присутствовали все элементы, образующие множество.

Пример 2.30. Задание списков и множеств

```
> [a,b,c],[a,c,b],[a,a,c,c,b,a]; # Задание разных списков
```

$$[a, b, c], [a, c, b], [a, a, c, c, b, a]$$

```
> {a,b,c},{a,c,b},{a,a,c,c,b,a}; # Задание одинаковых множеств
```

$$\{a, b, c\}, \{a, b, c\}, \{a, b, c\}$$

Списки, как и последовательности, сохраняют порядок своих элементов, поэтому с помощью индекса можно получить значение любого элемента списка, что, впрочем, справедливо и для множества. Более того, индексной форме элемента списка можно присвоить новое значение, изменив тем самым значение соответствующего элемента списка. Подобная техника не проходит для элементов множества.

Пример 2.31. Получение и изменение значений элементов списка и множества

```
> l:=[a,b,c];
```

$$l := [a, b, c]$$

```
> l[2];
```

$$b$$

```
> l[3]:=3; l;
```

$$l_3 := 3$$

$$[a, b, 3]$$

```
> s:={a,a,c,c,b,a};
```

$$s := \{a, b, c\}$$

```
> s[2];
```

$$b$$

```
> s[3]:=3; s;
Error, cannot assign to a set
```

$$\{a, b, c\}$$

Замечание

При создании множества порядок его элементов формируется системой Maple и может меняться от сеанса к сеансу при работе с одним и тем же рабочим листом.

При выборе нескольких элементов списка или множества с помощью индекса можно использовать объект диапазон, причем в этом случае положительные значения индекса соответствуют отсчету элементов в списке или множестве слева направо, а отрицательные значения соответствуют отсчету

справа налево. Например, если не известно количество членов в списке s, то все их можно выбрать командой:

```
> s[1..-1];
```

Вообще, при выборе элементов с помощью индекса его значение, равное -1, соответствует последнему элементу списка, -2 соответствует предпоследнему и т. д.

Чтобы изменить элемент множества, его следует удалить операцией minus, а затем добавить новый элемент операцией union, семантика которых соответствует их аналогам в математике: разности и объединению. Кроме этих двух операций в Maple реализована операция пересечения двух множеств intersect.

Пример 2.32. Операции со множествами

```
> ({a,b,c} minus {c}) union {3};
```

$$\{3, a, b\}$$

```
> {a,b,c} intersect {b,c,d};
```

$$\{b, c\}$$

Узнать, является ли некоторое выражение элементом списка или множества, можно командой member(), первым параметром которой следует задать проверяемое выражение, а вторым — имя переменной, в которой хранится список или множество:

```
> s:={x^2,x^(-2),x,1/x};
```

$$s := \{x, x^2, \frac{1}{x}, \frac{1}{x^2}\}$$

```
> member(x^(-1),s);
```

$$true$$

```
> member(1,s);
```

$$false$$

2.3.1.3. Массивы и таблицы

Массив является дальнейшим развитием концепции списка. Если список можно мыслить как перенумерованную последовательность, индекс которой может принимать только положительные значения, причем нумерация обязательно начинается с единицы, то в массиве каждый элемент также связан с индексом, однако не ограничен одной размерностью. Массив может иметь много размерностей, каждую со своим индексом. Более того, изменение индекса не ограничено положительными целыми значениями, его величина может быть как отрицательным целым, так и нулем.

Для объявления массива следует использовать функцию `array()` в правой части операции присваивания. Переменная в левой части будет представлять вновь созданный массив. Синтаксис функции создания массива следующий:

```
array(индексная_функция, границы, список)
```

Параметр `индексная_функция` должен быть именем процедуры, задающей, каким образом выполняется индексация (встроенные значения `symmetric`, `antisymmetric`, `sparse`, `diagonal` и `identity` позволяют задать симметричную, кососимметричную, разреженную, диагональную и единичную матрицы, см. страницу Справки, отображаемую командой `?indexfcn`). Параметр `границы` представляет диапазон(ы) изменения индекса(ов) массива. Если массив многомерный, то соответствующие диапазоны должны задаваться подряд через запятую. Значения элементов массива задаются параметром `список`, причем для двумерного массива элементами списка являются списки, содержащие значения соответствующих строк массива, т. е. этот параметр является списком списков. Для массивов большей размерности он представляет собой вложенные списки с глубиной, равной количеству размерностей. Все параметры этой функции являются не обязательными, однако, либо параметр, задающий границы, либо список значений должен обязательно присутствовать.

Если значения элементов массива не заданы, то с использованием индексной формы можно присвоить элементам массива соответствующие значения, причем в квадратных скобках следует задавать список индексов, соответствующих элементу, которому присваивается значение. Сразу же отметим, что когда создан массив и его элементам присвоены значения, то простой набор в области ввода рабочего листа имени массива не приведет к отображению его содержимого, а будет всего *лишь* напечатано имя массива. Для отображения в области вывода значений элементов массива следует воспользоваться командой `print()`, которая отображает в области вывода (не на принтере!) содержимое объекта, заданного в качестве ее параметра.

Пример 2.33. Задание массивов

```
> ar:=array(1..3);
```

$$ar := \text{array}(1..3, [\ ])$$

```
> ar[1]:=1: ar[2]:=2: ar[3]:=3:
> print(ar);
```

$$[1, 2, 3]$$

```
> ar0:=array(2..3,[2,3]);
```

$$ar0 := \text{array}(2..3, [$$
$$(2) = 2$$
$$(3) = 3$$
$$])$$

```
> ar1:=array([[1,2],[2,1]]);
```

$$ar1 := \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

```
> ar1[2,2]:=3; print(ar1);
```

$$ar1_{2,2} := 3$$

$$\begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$$

Обратите внимание, каким образом печатается массив, у которого индексы изменяются не от 1 и далее (массив ar0).

Таблица является дальнейшим развитием массива, как структуры данных. В ней в качестве индекса можно использовать не только целые числа, а все, что угодно. Для создания таблицы используется функция table(), параметрами которой являются индексная функция и список или множество пар индекс=значение.

Пример 2.34. Задание таблицы

```
> t:=table([one=1,two=2]);
```

$$t := \text{table}([one = 1, two = 2])$$

```
> t[two];
```

$$2$$

Таблицы достаточно удобный объект, когда надо в одном "массиве" хранить данные, относящиеся к какому-либо реальному объекту, и ссылаться к ним по индексам, представляющим естественную запись их наименований. Приведем в качестве примера таблицу, содержащую данные по физико-механическим характеристикам стали:

```
> steel:=table([mass=[7.8*10^3,kg/m^3],
               elasticity=[2.1*10^5,MPa]]);
```

$$steel := \text{table}([elasticity = [210000.0, MPa], mass = \left[7800.0, \frac{kg}{m^3}\right]])$$

```
> steel[elasticity];
```

$$[210000.0, MPa]$$

2.3.2. Структура выражений и работа с ней

Преобразуя алгебраические выражения в Maple, пользователь в основном оперирует ими как математическими объектами: извлекает корни, возводит в степени, вычисляет интегралы и производные и т. п. Однако часто возникает необходимость выделить из выражения его часть и именно с ней произвести некоторые преобразования, или заменить ее на некоторое другое

выражение. В таких случаях взгляд на алгебраические выражения, как математические объекты, не принесет никаких плодов, а вот знание структуры выражения, или как оно хранится и обрабатывается в системе Maple, т. е. внутреннее представление выражений, поможет в решении специфических задач преобразования выражений. В случае специфических выражений, состоящих только из списков или множеств, задача манипулирования их элементами достаточна проста и рассмотрена нами в предыдущем разделе 2.3.1, а вот как хранятся и каким образом можно получить доступ к составляющим частям общих алгебраических выражений — это задача более серьезная. Именно эти вопросы и будут предметом нашего рассмотрения в данном разделе, но начнем мы все-таки с обзора дополнительных возможностей Maple при работе со структурами списков, множеств и полиномов.

2.3.2.1. Структурная обработка списков, множеств и полиномов

Часто возникает необходимость выполнить какую-либо команду или вычислить функцию применительно к каждому элементу списка или множества. Можно, конечно, это сделать, поочередно выбирая элементы и применяя к каждому из них последовательно команду или функцию. Такой способ не совсем удобен, так как необходимо знать количество элементов в списке или множестве, да и к тому же уметь организовывать циклические вычисления. (О них мы еще не говорили, но язык Maple позволяет реализовывать подобные вычисления. Об этом, правда, разговор пойдет в гл. 5.) К счастью, разработчики Maple, предвидя подобные задачи, разработали несколько команд, действие которых распространяется на каждый элемент списка или множества, а не на весь список или множество как единое целое.

Команды `map()` и `map2()` позволяют применить функцию или команду, заданную первым параметром, ко всем элементам списка или множества, возвращая, соответственно, список или множество. Их общий синтаксис имеет вид:

```
map(функция, список | множество [, пар2, пар3, ..., парN]);
map2(функция, пар1, список | множество [, пар3, ..., парN]);
```

Если для выполнения команды или функции, заданной первым параметром команды `map()`, необходимы дополнительные параметры, то их следует задавать после списка или множества. Команда `map2()` отличается от команды `map()` тем, что элементы списка или множества передаются в качестве второго параметра команды или функции, определенной первым параметром.

Пример 2.35. Выполнение команд над элементами списка или множества

```
> map(int, [x,x^2,x^3], x);
```

$$\left[\frac{1}{2}x^2, \frac{1}{3}x^3, \frac{1}{4}x^4\right]$$

```
> map(x->x^a, {x,y,z});
```

$$\{x^a, y^a, z^a\}$$

```
> map2(diff, x^y/ln(z), [x,y,z]);
```

$$\left[\frac{x^y\,y}{x\ln(z)}, \frac{x^y\ln(x)}{\ln(z)}, -\frac{x^y}{\ln(z)^2\,z}\right]$$

Замечание

Аналогично спискам и множествам команды `map()` и `map2()` работают и с элементами массивов и таблиц, если они переданы в качестве соответствующих параметров этих команд.

Замечание

Можно вместо списков и множеств передавать в эти команды и общие алгебраические выражения. В этом случае они работают со всеми операндами структуры выражения, о которых речь пойдет ниже в разделе 2.3.3.

Нам известна команда `seq()` формирования последовательности, которая также поэлементно обрабатывает список или множество:

```
> seq(sin(i),i=[x,y,z]);
```

$$\sin(x), \sin(y), \sin(z)$$

В Maple существует еще две похожих на нее команды `add()` и `mul()`. Первая формирует сумму, а вторая — произведение элементов списка или множества:

```
> add(sin(i),i=[x,y,z]);
```

$$\sin(x) + \sin(y) + \sin(z)$$

```
> mul(sin(i),i=[x,y,z]);
```

$$\sin(x)\sin(y)\sin(z)$$

Замечание

Эти команды также работают и с операндами структуры общих алгебраических выражений.

Мы умеем выбирать элементы списка или множества с помощью индекса. Maple позволяет выбрать элементы, удовлетворяющие некоторому условию. Для этого следует прежде всего определить функцию, результатом выполнения которой будет булево значение `true` или `false` в зависимости от того, истинно или нет некоторое определенное в ней условие. Например, следующая функция `sq()` возвращает булево значение `true`, если квадрат ее аргумента больше `1`:

```
> sq:=x->is(x^2>1);
```

$$sq := x \to \mathrm{is}(1 < x^2)$$

Теперь можно воспользоваться командой `select()`, передав ей в качестве параметра имя булевой функции `sq`, а вторым параметром задать список/множество, из которого будут выбраны элементы, квадраты которых больше `1`, и представлены в виде списка/множества:

```
> l:=[1,Pi,exp(1),0];
```

$$l := [1, \pi, e, 0]$$

```
> select(sq, l);
```

$$[\pi, e]$$

Действие команды `remove()` противоположно действию команды `select()`. Она возвращает список/множество, состоящий из элементов, не удовлетворяющих условию булевой функции, имя которой определено первым параметром, из списка/множества, заданного вторым параметром:

```
> remove(sq, l);
```

$$[1, 0]$$

Выполнить обе операции одновременно позволяет функция `selectremove()`, которая возвращает последовательность двух списков, первый из которых представляет результат выполнения команды `select()`, а второй — команды `remove()`:

```
> selectremove(sq,l);
```

$$[\pi, e], [1, 0]$$

Использованная нами в примерах функция была без дополнительных параметров: только элементы списка/множества передавались этой функции в качестве единственного параметра. Если для булевой функции необходимы дополнительные параметры, то они задаются во всех трех представленных функциях после списка/множества. Так, можно было бы не создавать собственную функцию `sq()`, а воспользоваться непосредственно функцией `is()`:

```
> select(is,l,RealRange(Open(1),infinity));
```

$$[\pi, e]$$

Точно также можно выбрать с помощью функции `type()` (ее вторым параметром задается тип Maple) все числа из списка `l`:

```
> select(type,l,numeric);
```

$$[1, 0]$$

Замечание

Функции `select()`, `remove()` и `selectremove()` могут работать и с общими алгебраическими выражениями, осуществляя проверку условия над каждым его операндом.

Линейное объединение двух списков можно реализовать с помощью команды op(), которая возвращает последовательность элементов списка, переданного ей в качестве параметра, и квадратных скобок, формирующих список из последовательности:

```
> s1:=[Pi,exp(1)];s2:=[0,1];
```

$$s1 := [\pi, e]$$

$$s2 := [0, 1]$$

```
> s:=[op(s1),op(s2)];
```

$$s := [\pi, e, 0, 1]$$

Более сложные объединения списков реализуются командой zip(), имеющей следующий синтаксис:

```
zip(бинарная_функция, список1, список2 [, значение]);
```

Семантика этой команды такова: бинарная (двух аргументов) функция выполняется, последовательно используя в качестве своих параметров элементы двух списков, формируя новый список из вычисленных значений. Длина полученного списка равна длине наименьшего из двух списков, переданных этой команде в качестве параметров, если не задан четвертый необязательный параметр. В случае его задания он используется в качестве элементов списка наименьшей длины при продолжении последовательного выбора элементов списка большей длины. Таким образом, в этом случае команда zip() формирует список, длина которого равна длине наибольшего списка-параметра.

Пример 2.36. Объединение списков

```
> zip(gcd,[0,14,8],[2,6,12]); # Функция gcd() вычисляет наибольший общий
                              # делитель двух своих аргументов.
```

$$[2, 2, 4]$$

```
> zip((x,y)->x+y,[1,2,3],[4,5,6]);
```

$$[5, 7, 9]$$

```
> zip((x,y)->x+y,[1,2,3],[4,5],0);
```

$$[5, 7, 3]$$

Замечание

Команда zip() может объединять также матрицы и векторы одинаковой размерности. Описание этих объектов можно найти в разделе 3.2.1, описывающем пакет линейной алгебры linalg.

Списки и полиномы сохраняют порядок следования, соответственно, своих элементов и членов с момента их создания именно так, как пользователь их задавал. Иногда возникает задача перестроить их таким образом, чтобы эле-

менты или члены шли в некотором специальном порядке. Для этого в Maple существует команда `sort()`, которая упорядочивает элементы списка в возрастающем порядке, а члены полинома в убывающем порядке относительно степеней его переменной. Если список содержит только числовые элементы, то используется обычное числовое упорядочивание; если список содержит только строковые элементы или символьные имена, то упорядочивание осуществляется с использованием лексикографического упорядочивания; если список содержит смешанные элементы (числа, строки и алгебраические выражения), то упорядочивание происходит по адресам памяти, в которых располагаются его элементы. (В этом последнем случае результаты сортировки могут меняться от сеанса к сеансу.)

Пример 2.37. Простая сортировка списков и полиномов

```
> sort([c,a1,a, "b"]);
```

$$[a, a1, "b", c]$$

```
> sort([2,4,7,-2,10]);
```

$$[-2, 2, 4, 7, 10]$$

```
> sort([c,a1,a, "b", "b23"]);
```

$$[a, a1, "b", "b23", c]$$

```
> sort([34,x^2,c,"78x"]);
```

$$[34, c, "78x", x^2]$$

```
> p:=x^2+a*x+a^2+b^2+x^4;
```

$$p := x^2 + a\,x + a^2 + b^2 + x^4$$

```
> sort(p);
```

$$x^4 + x^2 + x\,a + a^2 + b^2$$

Внимание!

При сортировке полинома следует помнить, что он сортируется "по месту", т. е. после его сортировки он хранится в соответствии с выполненной сортировкой, тогда как сортировка списка не влияет на его хранение — он продолжает храниться в том порядке, как вводились его элементы при создании списка.

Вторым параметром команды `sort()` можно задать булеву функцию, определяющую алгоритм упорядочивания элементов списка, или некоторые специальные значения для изменения алгоритма упорядочивания числовых, строковых или символьных списков, используемого по умолчанию. Символ `` `<` `` (именно в обратных кавычках) и `numeric` соответствуют упорядочиванию числового списка в возрастающем порядке, а символ `` `>` `` — в убывающем порядке. Для строкового или символьного списка значения `lexorder` или `string` вызывают упорядочивание списка с использованием лексикографического порядка. Значение `address` соответствует упорядочиванию любого списка в соответствии с адресами областей памяти, в которых расположены элементы списка.

Пример 2.38. Сортировка списков в соответствии с заданным алгоритмом

```
> sort([1/2,3/4,1/7,5/2],(x,y)->evalb(denom(x)<denom(y)));
```

$$\left[\frac{5}{2}, \frac{1}{2}, \frac{3}{4}, \frac{1}{7}\right]$$

```
> sort([2,4,7,-2,10],`>`);
```

$$[10, 7, 4, 2, -2]$$

При работе с полиномами очень часто необходимо выделить коэффициент при соответствующей степени переменной. Команда `coeff()` позволяет выполнить эту работу:

```
> p:=z^2*5+a+b+z^2*(a^2+b)+x*6/7;
```

$$p := 5\,z^2 + a + b + z^2\,(a^2 + b) + \frac{6}{7}x$$

```
> coeff(p,z^2);
```

$$5 + a^2 + b$$

Замечание

Для выделения некоторых элементов полинома можно использовать также команды: `lcoeff()` для получения старшего коэффициента (при максимальной степени переменной), `tcoeff()` для выделения младшего коэффициента (при минимальной степени переменной), `coeffs()` для формирования последовательности всех коэффициентов полинома (в том порядке, как он задавался). Более подробную информацию об этих командах можно найти в справочной системе Maple.

2.3.2.2. Внутренняя структура выражений

Каждое алгебраическое выражение хранится системой Maple в виде древовидной структуры, обеспечивая тем самым доступ к любому ее члену или подвыражению, а также позволяя выполнять над ним разнообразные символьные преобразования. В представлении этой структуры каждый объект Maple, в том числе и выражение, делится на подобъекты первого уровня, которые, в свою очередь, также делятся на подобъекты и т. д. Этот процесс продолжается до тех пор, пока не будут получены базисные простые элементы Maple (объекты основных типов: целые, вещественные, дроби, неизвестные величины и т. д.). Но прежде чем знакомить читателя со структурой общего алгебраического выражения, мы остановимся на нескольких командах, позволяющих выделять части таких объектов, как уравнение, диапазон и дробь, в том числе алгебраическая.

Уравнение представляется в виде двух выражений, соединенных знаком равенства. Его не следует путать с операцией присваивания (:=), которая переменной в левой части присваивает значение выражения в правой части.

Уравнение является объектом Maple и служит для задания действительных математических уравнений. Его можно использовать в правой части операции присваивания, именуя тем самым уравнение. Для выделения левой части уравнения предназначена команда `lhs()`. Выделить его правую часть можно командой `rhs()`.

Пример 2.39. Задание уравнения и выделение его частей

```
> x^2+sin(x)=cos(x)-1;
```

$$x^2 + \sin(x) = \cos(x) - 1$$

```
> eq:=m*v^2/2+m*g*h=C;
```

$$eq := \frac{1}{2} m\, v^2 + m\, g\, h = C$$

```
> rhs(%%);
```

$$\cos(x) - 1$$

```
> lhs(eq);
```

$$\frac{1}{2} m\, v^2 + m\, g\, h$$

Эти же команды работают и с объектом диапазон, выделяя его начало и конец:

```
> lhs(4..8);
```

$$4$$

```
> rhs(4..8);
```

$$8$$

Для выделения числителя и знаменателя числовой или алгебраической дроби служат команды, соответственно, `numer()` и `denom()`, причем перед выделением этих частей дробей Maple осуществляет их упрощение, приводя к нормальной форме.

Пример 2.40. Выделение числителя и знаменателя дроби

```
> afrac:=(sin(x)+x/y)/(cos(x)/sin(x)+2);
```

$$afrac := \frac{\sin(x) + \dfrac{x}{y}}{\dfrac{\cos(x)}{\sin(x)} + 2}$$

```
> numer(afrac);
```

$$(\sin(x)\, y + x)\, \sin(x)$$

```
> denom(afrac);
```

$$y\, (\cos(x) + 2\, \sin(x))$$

```
> numer(4/5/6*34/7);
```

$$68$$

```
> denom(4/5/6*34/7);
```

$$105$$

Теперь обратимся к общему алгебраическому выражению и посмотрим, каким образом можно получить в Maple доступ к его структурному представлению через базовые элементы. Рассмотрим выражение:

```
> expr:=x^7/sin(x)+8*x^5+3*sqrt(x)*sin(x);
```

$$expr := \frac{x^7}{\sin(x)} + 8\,x^5 + 3\,\sqrt{x}\,\sin(x)$$

Воспользуемся командой `whattype()`, которая скажет нам, какой тип имеет наше выражение:

```
> whattype(expr);
```

$$+$$

Замечание

О всех допустимых типах Maple можно прочитать на странице Справки, которая отображается командой `?type`.

Как видим, наше выражение представляет собой сумму. Посмотрим теперь, что представляют собой члены этой суммы, или, по принятой в Maple терминологии, *операнды выражения*. Команда `nops(выражение)` определяет количество операндов выражения, а команда `op(выражение)` выдает их в виде последовательности выражений. Эта же команда позволяет извлечь конкретный операнд выражения, указав в качестве первого параметра его порядковый номер.

```
> nops(expr);
```

$$3$$

```
> op(expr);
```

$$\frac{x^7}{\sin(x)},\ 8\,x^5,\ 3\,\sqrt{x}\,\sin(x)$$

```
> op3:=op(3,expr);
```

$$op3 := 3\,\sqrt{x}\,\sin(x)$$

Выделенный третий операнд, как и предыдущие, является выражением, представляющим произведение трех членов. Используя функции `whattype()`, `nops()` и `op()`, мы точно также можем исследовать структуру третьего операнда:

```
> whattype(op3);
```

$$*$$

```
> nops(op3);
```

$$3$$

```
> op(op3);
```

$$3, \sqrt{x}, \sin(x)$$

```
> op3_2:=op(2,op3);
```

$$op3_2 := \sqrt{x}$$

Второй операнд третьего операнда нашего выражения является степенным выражением, имеющем два операнда, первый из которых является неизвестной величиной (тип `symbol`), а второй — дробь (тип `fraction`), которая, в свою очередь, имеет два целых операнда (тип `integer`):

```
> whattype(op3_2);
```

$$\wedge$$

```
> op(op3_2);
```

$$x, \frac{1}{2}$$

```
> op3_2_1:=op(1,op3_2);
```

$$op3_2_1 := x$$

```
> whattype(op3_2_1);
```

$$symbol$$

```
> op3_2_2:=op(2,op3_2);
```

$$op3_2_2 := \frac{1}{2}$$

```
> whattype(op3_2_2);
```

$$fraction$$

```
> op(op3_2_2);
```

$$1, 2$$

```
> whattype(%[1]); whattype(%[2]);
```

$$integer$$
$$integer$$

Таким образом, исследуя только один операнд исходного выражения, мы дошли до операндов базовых типов `symbol` и `integer`, которые являются простыми и представляют самих себя. Исследование других операндов также завершится приходом к базовым простым типам. Суммируя проделанную работу по выявлению операндов разных уровней исходного выражения, его можно представить в виде дерева выражения, в узлах которого располагаются типы выражений и подвыражений, а ветви представляют соответствующие их операнды. На рис. 2.1 показано дерево нашего исходного выражения.

Операндами списка или множества являются его элементы:

```
> op([1,4,6,8]);
```

$$1, 4, 6, 8$$

```
> op({x^2,x^3});
```

$$x^2, x^3$$

Другие структурированные объекты (массивы, таблицы, матрицы, векторы) имеют более сложное внутреннее представление в Maple и не представляются таким простым одноуровневым деревом, как списки и множества. В качестве упражнения вы можете исследовать структуру массива, придерживаясь описанной выше технологии.

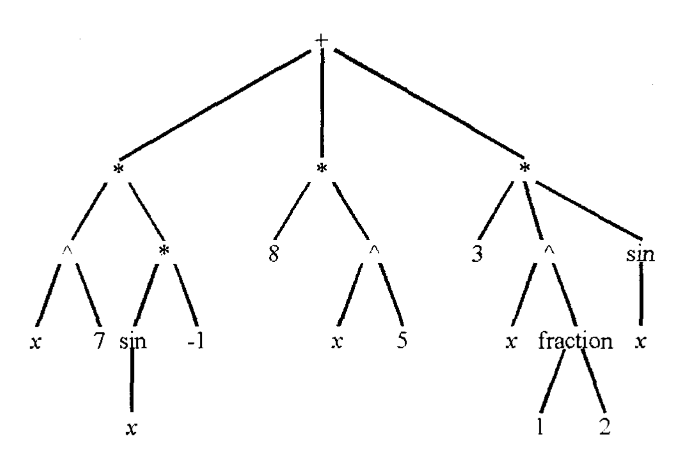


Рис. 2.1. Дерево выражения

Если читатель помнит, то рассказывая о командах, выделяющих элементы из списка/множества (`select()`, `remove()`), или команде `map()`, применявшей функции последовательно ко всем элементам списка/множества, мы в замечаниях отмечали, что они работают и с общими алгебраическими выражениями. Действительно, семантика этих команд остается такой же, как и при работе со списками/множествами, но в отличие от использования в них упомянутых объектов, предоставлявших свои элементы для выполнения последовательности однотипных действий, при использовании алгебраических выражений итерации осуществляются по операндам этих выражений. Например, следующая команда `map()` вычислит квадрат каждого операнда выражения и возвратит результат в виде исходного выражения, но с другими операндами:

```
> map(x->x^2,y^m);
```

$$(y^2)^{(m^2)}$$

В выражении этого примера, имеющем экспоненциальный тип `^`, два операнда `y` и `m`, каждый из которых возводится в квадрат, а потом измененный первый операнд возводится в степень измененного второго операнда.

Аналогичным образом работают и команды `select()` и `remove()`:

```
> l := x -> evalb(is(x<0)=true);
```

$$l := x \to \mathbf{evalb}(\mathrm{is}(x < 0) = \mathit{true})$$

```
> select(1, sin(x)-5-cos(x)-x^2);
```

$$-5$$

```
> remove(type, sin(x)-5-cos(x), function);
```

$$-5 - \cos(x)$$

Команда `remove()`, казалось бы, должна удалить все функции из выражения. Однако этого не произошло?! Эта команда в нашем примере должна удалить из выражения все операнды, имеющие тип `function`. Проверим, какой тип имеет последний третий операнд:

```
> whattype(op(3,sin(x)-5-cos(x)));
```

$$*$$

```
> op(op(3,sin(x)-5-cos(x)));
```

$$-1, \cos(x)$$

Оказывается, тип третьего операнда произведение `*`, а не функция, как мы думали, поэтому-то член `cos(x)` и не был удален из выражения. Дело в том, что тип всего выражения сумма `+`, и поэтому тип третьего операнда не функция, а произведение, так как функцию `cos(x)` следует умножить на `-1`.

Maple предлагает большое количество булевых функций, которые можно использовать в командах `select()` и `remove()` для работы со структурой выражений. Мы только расскажем об их небольшом числе, остальные можно найти в справочной системе Maple.

Команда `has()` определяет, содержится ли некоторое подвыражение в заданном выражении.

Пример 2.41. Функция has()

```
> has(x*exp(cos(x+2)),x+2);
```

true

```
> has(x*exp(cos(x+2)),cos);
```

true

```
> select(has, cos(x)+sin(x)+cos(2*x)*sin(x), cos);
```

$$\cos(x) + \cos(2\,x)\sin(x)$$

```
> remove(has, exp(cos(x))+sin(x)+cos(2*x)*sin(x)+exp(x+y), exp);
```

$$\sin(x) + \cos(2\,x)\sin(x)$$

Команда `has()` понимает только те подвыражения, которые могут быть определены с помощью команды `op()` при разборе структуры выражения. Поэтому если необходимо выделить из выражения только члены, содержащие некоторую функцию, то в команде `has()` следует задавать лишь имя этой функции, как показано в примере 2.41.

Замечание

В функции has() можно задать несколько подвыражений в виде списка. Ее результатом будет истина тогда и только тогда, когда найдено хотя бы одно из подвыражений в списке.

Команда hastype() определяет, содержит ли выражение подвыражения заданного типа:

```
> select(hastype, exp(cos(x))+sin(x)+cos(2*x)*sin(x)+exp(x*y), `*`);
```

$$\cos(2\,x)\sin(x)+e^{(x\,y)}$$

```
> select(hastype, exp(cos(x))+sin(x)+cos(2*x)*sin(x)+exp(x*y), `+`);
```

$$0$$

Обратите внимание, если в выражении не найдено ни одно подвыражение заданного типа, то функция select() возвращает 0.

Если необходимо выделить из выражения не операнды, содержащие подвыражения заданного типа, а сами подвыражения, то следует использовать команду indets(), вторым параметром которой задается тип подвыражения:

```
> indets(exp(cos(x))+sin(x)+cos(2*x)*sin(x)+exp(x*y), `*`);
```

$$\{\cos(2\,x)\sin(x), x\,y, 2\,x\}$$

Эта функция возвращает в виде множества все подвыражения указанного типа.

Не все подвыражения можно выделить, используя только типы данных, поддерживаемые Maple. Дело в том, что некоторые, встречающиеся в математике операторы, не имеют соответствующего типа. Например, операция дифференцирования. В этом случае следует использовать вместо типа Maple специальную функцию specfunc(type,name), которая связывает имя оператора name с типом type. Следующий пример демонстрирует выделение из выражения членов с операцией дифференцирования:

```
> DE:=expand(diff(sin(y(t))*t^2,t));
```

$$DE := \cos(\mathrm{y}(t))\left(\frac{\partial}{\partial t}\mathrm{y}(t)\right)t^2+2\sin(\mathrm{y}(t))\,t$$

```
> select(hastype,DE,specfunc(anything,diff));
```

$$\cos(\mathrm{y}(t))\left(\frac{\partial}{\partial t}\mathrm{y}(t)\right)t^2$$

2.3.2.3. Подстановка и преобразование типов

При выполнении математических преобразований часто необходимо произвести замену переменных в выражении, функции, уравнении и т. д., то есть вместо какой-то переменной подставить ее представление через некоторые

другие переменные. Для этих целей в Maple существует команда subs(), синтаксис которой имеет следующий вид:

```
subs( старое_выражение = новое_выражение, выражение);
subs( s1, ..., sn, выражение);
```

Во второй форме этой команды каждое из s1, ..., sn является уравнением или списком/множеством уравнений.

Первая форма команды анализирует выражение, выделяет в нем все вхождения старое_выражение и подставляет вместо них новое_выражение. Вторая форма позволяет выполнить серию подстановок в выражение. Подстановки выполняются последовательно, начиная с s1. Это означает, что после выполнения первой подстановки, определенной уравнением s1, Maple отыскивает вхождения левой части уравнения s2 во вновь полученном выражении и заменяет каждое такое вхождение на выражение, заданное в левой части уравнения s2. Если подстановки заданы в виде списка или множества уравнений, то они выполняются одновременно, т. е. вхождения выражений, заданных в левых частях уравнений, определяются в исходном параметре выражение. Например, после выполнения подстановки subs(x=y, y=x, [x,y]) исходный список [x,y] будет преобразован в [x,x], тогда как при использовании команды subs({x=y, y=x}, [x,y]) переменные x и y в списке поменяются местами: список будет иметь вид [y,x].

Пример 2.42. Подстановки в выражении

```
> ex:=cos(x)+cos(x)^(1/3);
```

$$ex := \cos(x) + \cos(x)^{(1/3)}$$

```
> subs(cos(x)=27,ex);
```

$$27 + 27^{(1/3)}$$

```
> simplify(%);
```

$$30$$

```
> ex1:=s^3;
```

$$ex1 := s^3$$

```
> subs(s^2=1-c^2,ex1);
```

$$s^3$$

Последняя команда подстановки subs() в примере 2.42 не подставила в выражение s^3 вместо s^2 выражение $1-c^2$. Дело в том, что эта команда осуществляет замену, только если левая часть уравнения подстановки совпадает с одним из операндов в структурном представлении выражения. Такая подстановка называется "синтаксической подстановкой".

Для выхода из подобных ситуаций можно предложить несколько способов. Первый заключается в том, что следует явно выразить переменную s из

уравнения $s^2 = 1 - c^2$ и снова воспользоваться командой лексической подстановки subs():

```
> subs(s=sqrt(1-c^2),ex1);
```

$$(1 - c^2)^{(3/2)}$$

Можно воспользоваться командой simplify(), указав в ней в качестве параметра требуемую замену:

```
> simplify(ex1,{s^2=1-c^2});
```

$$s - s\,c^2$$

И, наконец, можно воспользоваться командой algsubs(), осуществляющей не синтаксическую, а алгебраическую подстановку:

```
> algsubs(s^2=1-c^2,ex1);
```

$$s\,(1 - c^2)$$

Обратите внимание, что в первом варианте в выражении заменены все вхождения переменной s, тогда как в двух других эта переменная остается в результирующем выражении.

Если известно, какой операнд выражения необходимо заменить, то следует использовать команду subsop() со следующим синтаксисом:

```
subsop(уравнение1, уравнение2, ..., уравнениеn, выражение);
```

Первые параметры представляют уравнения, в правой части которых стоят порядковые номера операндов выражения, заданного последним параметром выражение, а правые части представляют выражения, на которые заменяются соответствующие операнды:

```
> ex:=cos(x)+cos(x)^(1/3);
```

$$ex := \cos(x) + \cos(x)^{(1/3)}$$

```
> subsop(1=x^2,ex);
```

$$x^2 + \cos(x)^{(1/3)}$$

```
> subsop([2,1,0]=sin,ex);
```

$$\cos(x) + \sin(x)^{(1/3)}$$

Обратите внимание на последний оператор подстановки. Здесь в правой части уравнения, задающей номер операнда, стоит список, в котором целые числа представляют порядковые номера операндов последующих уровней в структуре выражения. В нашем примере 2 соответствует второму операнду исходного выражения ($\cos(x)^{(1/3)}$), 1 — первому операнду этого второго операнда ($\cos(x)$), а 0 используется для имени функции в первом операнде второго операнда исходного выражения, который имеет тип функция. Таким способом можно в выражении заменять имена функций.

Иногда необходимо выполнить преобразование выражения одного типа в другой тип. Такое преобразование типов может потребоваться для выполнения некоторых действий над выражением с помощью команды, не работающей с исходным типом выражения. Например, нельзя строить график ряда Тейлора какой-либо функции, но всегда можно построить график полинома. Следовательно, следует выражение, имеющее тип `series` (ряд), преобразовать в выражение, имеющее тип `polynom` (полином), а потом воспользоваться командой `plot()` для построения графика.

Преобразовать выражение в другой тип можно командой `convert()`, первым параметром которой задается выражение, а вторым — тип, в который это выражение следует преобразовать.

Пример 2.43. Преобразование выражений

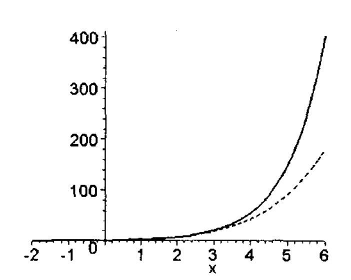
```
> t:=taylor(f,x=0);
```

$$t := 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + O(x^6)$$

```
> p:=convert(t,polynom);
```

$$p := 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5$$

```
> plot([f,p],x=-2..6,color=black,linestyle=[1,4]);
```



Преобразование типов в Maple имеет некоторые ограничения, т. е. нельзя преобразовать выражение произвольного типа в выражение другого типа, который поддерживается системой аналитических вычислений. Например, мы видели, что ряд можно преобразовать в полином, однако, обратное неверно:

```
> convert(p, series);
Error, unable to convert
Ошибка, невозможно преобразовать
```

Все допустимые преобразования можно посмотреть в справке по команде `convert()`.

2.3.3. Вычисление выражений

В программах символьных вычислений пользователь сталкивается с понятием *вычисление* символьных имен, которое отсутствует в системах численных расчетов. Действительно, так как можно использовать неопределенные переменные, или неизвестные, то какому значению должна равняться переменная `x`, если сначала ей присвоили неизвестную величину `y`, затем неизвестной `y` присвоили неизвестную величину `z`, и, наконец, переменной `z` присвоили число 5. Ответ на этот вопрос дают правила вычислений, включающие в себя понятие уровней вычислений.

2.3.3.1. Уровни вычислений

В большинстве случаев Maple использует алгоритм *полного вычисления* имен. Это означает, что когда необходимо вычислить значение символьной переменной, проверяется, присваивалось ли ей какое-либо значение. Если да, то оно подставляется вместо имени переменной и проверяется, содержит ли подставленное значение неизвестные переменные. Если содержит, то проверяется, было ли присваивание для этих имен и процесс продолжается дальше рекурсивно, пока вместо имен всех переменных не будут подставлены присвоенные им значения, или если им ничего не присваивалось, то такие имена останутся в окончательном результате вычисления имени переменной как неизвестные величины.

Теперь понятно, как Maple решит поставленную в начале данного подраздела задачу — значение переменной `x` будет равно 5.

Пример 2.44. Полное вычисление имени

```
> x:=y;
```

$$x := y$$

```
> y:=z;
```

$$y := z$$

```
> z:=5;
```

$$z := 5$$

```
> x;
```

$$5$$

При вычислении значения имени или символа может оказаться, что необходимо осуществить несколько подстановок, как в нашем примере. Каждая подстановка при вычислении имени называется *уровнем вычисления* имени, и все они последовательно нумеруются, начиная с самого первого присваивания значения имени.

Команда `eval()` служит для явного полного вычисления имени, заданного в качестве значения ее первого параметра. Вторым параметром, принимающем целые значения, можно задать глубину вычисления имени, определив

уровень, до которого имя следует вычислить. Пример 2.45 иллюстрирует использование этой команды для вычисления имени x, определенного в предыдущем примере 2.44.

Пример 2.45. Уровни вычисления имени

```
> eval(x);    # Полное вычисление
```

$$5$$

```
> eval(x,1); # Вычисление до первого уровня
```

$$y$$

```
> eval(x,2); # Вычисление до второго уровня
```

$$z$$

```
> eval(x,3); # Вычисление до третьего уровня
```

$$5$$

Правилу полного вычисления подчиняются переменные всех типов за исключением переменных, содержащих массивы, таблицы, матрицы и процедуры. Для них действует правило, в соответствии с которым они вычисляются до значения последнего присвоенного имени, содержащего указанные выше объекты. Такое правило для массивов, таблиц, матриц и процедур введено для того, чтобы сохранять компактное представление элементов массивов с не присвоенными значениями и не вычисленными командами, например $\sin(x)$. Для полного вычисления имен, хранящих массивы, таблицы и процедуры, следует явно инициировать полное вычисление командой `eval()`.

Пример 2.46. Уровни вычисления имени

```
> x:=y;
```

$$x := y$$

```
> y:=array([[1,2],[2,1]]);
```

$$y := \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

```
> x;
```

$$y$$

```
> eval(x);
```

$$\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

```
> eval(x,1);
```

$$y$$

```
> eval(x,2);
```

$$\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

По умолчанию Maple не отображает код библиотечных процедур при их полном вычислении функцией `eval()`. Такой режим отображения можно изменить, установив опцию `verboseproc`, равной 2, в команде `interface()`:

```
> eval(sin);
```

proc (*x*: *algebraic*) ...**end proc**

```
> interface(verboseproc=2);
> eval(exp);
```

```
proc (x::algebraic )
local  i, t, q, n, f, r;
option  `Copyright (c) 1992 by the University of Waterloo. All rights reserved.`          ;
    if nargs ≠ 1 then  error  "expecting 1 argument, got %1"     , nargs
    elif  type( x, 'complex (float )') then  evalf( 'exp'( x ) )
    . . . . . . . . . . . . . . . . . . . . .
        end if
    elif typematch ( −I×x, `&*` ( r::'rational ', f::'arctrig ' ) ) then  exp( x ) := exp( I×f)^r
    else  exp( x ) := 'exp'( x )
    end if
end proc
```

Замечание

В приведенном примере текст процедуры вычисления синуса приведен не полностью в связи с его большими размерами.

Иногда необходимо просто вычислить имя переменной, не инициируя алгоритм полного или частичного вычисления. Для подобных целей в Maple присутствует команда `evaln()`, которая "вычисляет имя" символьной переменной, переданной ей в качестве параметра, даже если этой переменной были присвоены значения:

```
> x:=u;
```

$$x := u$$

```
> x;
```

$$u$$

```
> evaln(x);
```

$$x$$

Если заключить имя переменной в одинарные кавычки, то их действие будет аналогично действию команды `evaln()`. Подобное обстоятельство часто используется для передачи в некоторые команды, которые в качестве параметра требуют имя переменной, "закавыченное" имя переменной, дабы избежать появления ошибок:

```
> i:=5;
```

$$i := 5$$

```
> sum(i^2,i=1..3);
Error, (in sum) summation variable previously assigned, second argument
evaluates to 5 = 1 .. 3

> sum('i'^2,'i'=1..3);
```

$$14$$

Если переменной присвоить ее же имя, заключенное в кавычки, то эта операция приведет к тому, что переменная снова станет "чистой", т. е. будут аннулированы все предыдущие присваивания, и ее снова можно использовать в качестве неизвестной величины:

```
> x:=1;
```

$$x := 1$$

```
> x;
```

$$1$$

```
> x:='x';
```

$$x := x$$

```
> x;
```

$$x$$

Действие операции заключения в одинарные кавычки распространяется на целые выражения, откладывая их полное вычисление:

```
> y:=1;
```

$$y := 1$$

```
> x:=y+1;
```

$$x := 2$$

```
> x:='y+1';
```

$$x := y + 1$$

Команда `assigned()` проверяет, было ли переменной, определенной ее параметром, присвоено какое-либо значение:

```
> assigned(i);
```

true

Задача вычисления имен — это специфическая задача систем аналитических вычислений, а в практике математических вычислений часто встречается задача вычисления значения некоторого выражения или функции в определенной точке. В Maple ее можно решить несколькими способами.

Например, присвоить неизвестной величине необходимое значение, и при очередном вычислении вместо этой величины будет подставлено ее значение и результатом будет значение выражения в заданной точке:

```
> g:=x^2+x+1;
```

$$g := x^2 + x + 1$$

```
> x:=1;
```

$$x := 1$$

```
> g;
```

$$3$$

При таком подходе, однако, мы "теряем" выражение как выражение с неизвестной переменной. Поэтому в подобных случаях лучше воспользоваться функцией `eval()`, задав ее вторым параметром уравнение, в левой части которого стоит неизвестная величина, а в правой — ее значение:

```
> g:=x^2+x+1;
```

$$g := x^2 + x + 1$$

```
> eval(g,x=1);
```

$$3$$

```
> g;
```

$$x^2 + x + 1$$

Переменная, хранящая выражение с неизвестной, продолжает его хранить и после вычисления командой `eval()`. Подобного же эффекта можно достичь и с помощью функции `subs()`, но использование функции `eval()` приводит к корректным математическим вычислениям. Эта функция умеет вычислять интегралы и производные, а также работать с кусочно-непрерывными функциями, тогда как использование для целей вычисления выражения в точке функции `subs()` может приводить к неправильным математическим результатам:

```
> der := diff(f(x),x) + f(x);
```

$$der := \left(\frac{\partial}{\partial x}\mathrm{f}(x)\right) + \mathrm{f}(x)$$

```
> eval(der,x=0);
```

$$\left.\left(\frac{\partial}{\partial x}\mathrm{f}(x)\right)\right|_{x=0} + \mathrm{f}(0)$$

```
> subs(x=0,der);
```

$$\mathrm{diff}(\mathrm{f}(0), 0) + \mathrm{f}(0)$$

Замечание

При использовании команды `eval()` для вычисления значения выражения в точке можно через запятую задавать значения для нескольких неизвестных или параметров:

```
> eval(a*x^2+b*x+c,x=1,a=2,b=3,c=0);
```

Maple по умолчанию стремится производить все преобразования и вычисления с максимальной точностью, т. е. там, где это возможно, использовать

арифметику с дробными числами и радикалами. Иногда это не совсем удобно, и необходимо получить результат в форме привычных десятичных чисел. Для вычисления и преобразования коэффициентов выражений, представленных обыкновенными дробями, существует команда `evalf()`, аппроксимирующая дробные числа десятичными числами, а также вычисляющая выражение и константы Maple в форме числа с плавающей точкой.

По умолчанию используется арифметика чисел с плавающей точкой с 10 значащими цифрами. Это значение можно изменить, переустановив системную константу `Digits` в другое значение, например:

```
> Digits:=15;
```

После выполнения этого оператора все последующие в сеансе Maple вычисления с плавающей точкой будут производиться с 15 значащими цифрами. Обратим внимание, что имя этой системной переменной начинается с прописной буквы.

Синтаксис команды `evalf()` следующий:

```
evalf(f);
evalf(f, n);
```

где `f` — вычисляемое выражение, а `n` — число значащих цифр, используемых при вычислении.

Пример 2.47. Вычисление выражений в десятичной форме

```
> evalf(Pi, 20);
```

$$3.1415926535897932385$$

```
> Digits:=30;
```

$$Digits := 30$$

```
> evalf(5/3*exp(-2)*sin(Pi/4));
```

$$.159494160850684873267979942007$$

```
> evalf(cos(1));
```

$$.540302305868139717400936607443$$

```
> evalf(3/4*x^2+1/3*x-sqrt(2), 10);
```

$$.7500000000\, x^2 + .3333333333\, x - 1.414213562$$

Для ускорения вычисления больших и сложных числовых выражений можно воспользоваться командой `evalhf()`, единственным параметром которой является вычисляемое выражение. Эта команда, в отличие от `evalf()`, использует вещественную арифметику процессора компьютера, а не программную эмуляцию. Правда, точность вычислений в этом случае зависит от разрядности компьютера и на машинах с 32-разрядным процессором не превышает 15 значащих цифр в мантиссе представления числа.

Замечание

Кроме перечисленных команд вычисления выражения в точке Maple предлагает специальные команды вычисления комплексных выражений `evalb()`, матриц `evalm()`, вычисления над разнообразными полями чисел `evala()` и использование арифметики интервалов `evalr()`. Со всеми этими командами читатель может познакомиться через справочную систему Maple.

2.4. Решение уравнений, неравенств и их систем

Практически ни одна задача не обходится без решения какого-нибудь уравнения или системы уравнений, неравенства или системы неравенств. Команда `solve()` системы Maple является универсальным средством, позволяющим решать алгебраические уравнения, неравенства и их системы. Но прежде чем перейти к синтаксису этой команды, следует еще немного остановиться на тех объектах, с которыми эта команда работает, — на уравнениях и неравенствах.

Как мы уже определили ранее, два выражения, соединенные знаком равенства (=), представляют самостоятельный тип данных Maple — *уравнение* (`equation`). Уравнения можно присваивать обычным переменным Maple, с ними можно осуществлять преобразования, используя обычные арифметические действия, которые выполняются отдельно для левой и правой частей уравнений. Эти действия позволяют преобразовать уравнения к виду, удобному для использования, а иногда и облегчающему Maple поиск решения. В примере 2.48 приведены некоторые преобразования, которые можно осуществлять с уравнениями в Maple.

Пример 2.48. Допустимые операции с уравнениями

```
> 2*x^2+5=x+x^4;
```

$$2x^2+5=x+x^4$$

```
> whattype(%);
```

$$=$$

```
> f:=2*x^2+5=x+x^4;
```

$$f:=2x^2+5=x+x^4$$

```
> whattype(f);
```

$$=$$

```
> eq1:=sin(x)+cos(x)=cos(x)^2;
```

$$eq1:=\sin(x)+\cos(x)=\cos(x)^2$$

```
> eq1-(cos(x)=cos(x));
```

$$\sin(x)=\cos(x)^2-\cos(x)$$

```
> eq1+(cos(x)=cos(x));
```

$$\sin(x) + 2\cos(x) = \cos(x)^2 + \cos(x)$$

При проверке типа переменной, значением которой является уравнение, с помощью команды `whattype()` результатом является равенство =, означающее, что тип проверяемой переменной является уравнением.

Точно так же, как и при задании уравнений, два выражения, соединенные знаками >= (больше или равно), <= (меньше или равно), > (больше) или < (меньше), представляют новый тип — *неравенство* (`inequation`).

Пример 2.49. Неравенства

```
> x<y;
```

$$x < y$$

```
> whattype(%);
```

$$<$$

```
> f:=x>y;
```

$$f := y < x$$

```
> whattype(f);
```

$$<$$

```
> f-(z>4);
```

$$y - z < x - 4$$

```
> f-(z<4);
```

$$y - 4 < x - z$$

При проверке типа объекта, представляющего неравенство, в области вывода отображается либо <>, либо <, либо <=. Дело в том, что Maple "понимает" только эти три типа. Неравенства противоположного знака приводятся к ним перестановкой левой и правой частей с заменой знаков на противоположные.

2.4.1. Команда *solve()*

Команда `solve()` одна из самых полезных команд системы аналитических вычислений Maple, позволяющая решать уравнения и системы уравнений, неравенства и системы неравенств. Она всегда пытается найти замкнутое решение в аналитической форме. Ее синтаксис, как и синтаксис всех команд Maple, достаточно прост и легко запоминается:

```
solve(уравнение, переменная);
solve({уравнение1,уравнение2,...},{переменная1, переменная2,...});
```

Первая форма команды предназначена для решения одного уравнения относительно заданной переменной, тогда как вторая форма позволяет решать

системы уравнений относительно переменных, заданных вторым параметром. Обратим внимание на то, что система уравнений и ее неизвестные переменные задаются в виде множеств. Результатом в этом случае является также множество значений неизвестных в виде уравнений, тогда как в случае задания одного уравнения результатом будет выражение (в случае одного корня уравнения) или последовательность выражений (в случае нескольких корней). Если не задана переменная/переменные, относительно которых следует решать уравнение/систему уравнений, то Maple выдаст все решения относительно всех неопределенных переменных в исходных уравнениях. Если вместо уравнения задано выражение с неизвестными, то оно рассматривается как левая часть уравнения, тогда как правая часть предполагается равной 0. Пример 2.50 иллюстрирует некоторые из перечисленных ситуаций.

Пример 2.50. Решение уравнений и систем уравнений

```
> eq:=x^2-2*x+y^2=0;
```

$$eq := x^2 - 2\,x + y^2 = 0$$

```
> solve(eq,x);
```

$$1 + \sqrt{1 - y^2},\ 1 - \sqrt{1 - y^2}$$

```
> solve({eq},x);
```

$$\{x = 1 + \sqrt{1 - y^2}\},\ \{x = 1 - \sqrt{1 - y^2}\}$$

```
> eq1:=x+y=0;
```

$$eq1 := x + y = 0$$

```
> solve({eq,eq1},{x,y});
```

$$\{y = 0, x = 0\},\ \{x = 1, y = -1\}$$

```
> solve(eq);
```

$$\{y = \sqrt{-x^2 + 2\,x}, x = x\},\ \{y = -\sqrt{-x^2 + 2\,x}, x = x\}$$

Когда Maple не может найти ни одного решения, то команда `solve()` возвращает пустую последовательность `NULL`. Это означает, что либо решения не существует, либо Maple не удалось его найти. Если не удалось найти все решения, то глобальная переменная `_SolutionsMayBeLost` устанавливается равной `true`.

Решение последнего уравнения из примера 2.50 осуществлялось без указания переменной, относительно которой следовало решать уравнение. Maple решил их относительно всех неизвестных величин, входящих в уравнение. Причем он выбрал неизвестную x в качестве параметра (x=x), а неизвестную переменную y выразил через введенный параметр x. Чтобы получить решение, следует параметру x присвоить произвольное значение, тогда значение неизвестной y будет определено однозначно.

В общем случае полиномиальное уравнение степени выше 4 может не иметь решения, выраженного с помощью радикалов. В этом случае для представ-

ления результатов Maple использует специальную функцию `RootOf()`, которая применяется для обозначения любого корня выражения, заданного в качестве ее параметра:

```
> eq:=x^5+x^3+1=0;
```

$$eq := x^5 + x^3 + 1 = 0$$

```
> s:=solve(eq,x);
```

$$s := \mathrm{RootOf}(_Z^5 + _Z^3 + 1, index = 1),$$
$$\mathrm{RootOf}(_Z^5 + _Z^3 + 1, index = 2),$$
$$\mathrm{RootOf}(_Z^5 + _Z^3 + 1, index = 3),$$
$$\mathrm{RootOf}(_Z^5 + _Z^3 + 1, index = 4),$$
$$\mathrm{RootOf}(_Z^5 + _Z^3 + 1, index = 5).$$

```
> evalf(s[1]);
```

$$.6366631068 + .6647015651\ I$$

```
> solve(x=cos(x));
```

$$\mathrm{RootOf}(_Z - \cos(_Z))$$

В этом примере функция $\mathrm{RootOf}(_Z - \cos(_Z))$ представляет любое решение уравнения $_Z - \cos(_Z) = 0$. Обратим внимание на переменную $_Z$. Это системная переменная, сгенерированная Maple, которая всего лишь заменяет переменную x нашего уравнения. Опция `index` со значением, равным целому числу, служит для нумерации и упорядочивания корней уравнения. Обратим внимание, что с помощью функции `evalf()` можно получить приближенные числовые значения функции `RootOf`.

Особо отметим решение с помощью команды `solve()` тригонометрических уравнений. По умолчанию Maple решает их на промежутке $[-\pi, \pi]$. Для получения всех решений тригонометрических уравнений следует задать значение глобальной переменной `_EnvAllSolutions` равным `true`. Следующий пример иллюстрирует использование глобальной переменной `_EnvAllSolutions`:

```
> eq:=sin(x)^2+2*sin(x)+1=0;
```

$$eq := \sin(x)^2 + 2\sin(x) + 1 = 0$$

```
> s:=solve(eq,x);
```

$$s := -\frac{1}{2}\pi$$

```
> _EnvAllSolutions:=true;
```

$$_EnvAllSolutions := true$$

```
> s:=solve(eq,x);
```

$$s := -\frac{1}{2}\pi + 2\pi\ _Z1\sim$$

Как видно, в случае `_EnvAllSolutions:=true` Maple действительно строит все решения тригонометрического уравнения с использованием целочисленной системной переменной *_Z1~*, в которой знак тильда (~) означает, что на значения переменной наложены некоторые ограничения. В данном случае эта переменная может принимать только целочисленные значения. (В этом можно убедиться, выполнив команду `about(_Z1)`.) Подобные переменные используются Maple для представления всех решений тригонометрических уравнений. Префикс *_Z* в имени переменной, сгенерированной Maple, служит указанием того, что эта переменная может принимать только целые значения. Кроме указанных переменных также используются переменные с префиксом *_NN*, принимающие неотрицательные целые значения, и префиксом *_B*, для представления переменных с двоичной областью значений (0 или 1).

Замечание

Решение любого трансцендентного уравнения, в том числе и тригонометрического, достаточно сложная и серьезная проблема для систем аналитических вычислений. Иногда простое трансцендентное уравнение может и не решаться в Maple. В таких случаях следует всегда помнить о том, что Maple использует алгоритмический подход для решения уравнений, и, возможно, ему следует помочь, сделав кое-какие не стандартные преобразования уравнения, приведя его к другому виду.

Когда мы вручную решаем уравнения или системы уравнений, то после получения ответа мы всегда осуществляем проверку полученного решения, подставляя его в исходное уравнение или систему. Точно также следует поступать и при работе в Maple. И решения, получаемые в виде последовательностей множеств, когда уравнение или система уравнений заданы в виде множеств, оказываются самыми подходящими для этих целей.

Наиболее удобным и подходящим методом проверки решений является использование функции `eval()`. Пусть мы решили, например, систему уравнений:

```
> eqns:={x+2*y=3,y+1/x=1};
```

$$eqns := \{x + 2\,y = 3, y + \frac{1}{x} = 1\}$$

```
> sols:=solve(eqns,{x,y});
```

$$sols := \{x = -1, y = 2\}, \{x = 2, y = \frac{1}{2}\}$$

Последовательность множеств, представляющих два полученных решения, сохранена в переменной `sols`. Теперь, чтобы проверить правильность полученных решений, следует подставить их в исходную систему и вычислить полученные выражения с помощью команды `eval()`:

```
> eval(eqns,sols[1]);
```

$$\{1 = 1, 3 = 3\}$$

```
> eval(eqns,sols[2]);
```

$$\{1 = 1, 3 = 3\}$$

Как видим, в результате вычисления системы уравнений на двух полученных решениях мы получили тождества, что говорит о правильности наших решений. Если нам для дальнейших вычислений необходимо иметь значения первого решения в виде отдельных переменных, то той же самой командой `eval()` можно извлечь их, вычислив, соответственно, неизвестную x и y на первом решении:

```
> x1:=eval(x,sols[1]);
```

$$x1 := -1$$

```
> y1:=eval(y,sols[2]);
```

$$y1 := \frac{1}{2}$$

Для проверки решения можно использовать функцию `map()` вместе с функцией `subs()`, которая за одну операцию выполнит проверку всех решений. Это методика достаточно удобна, особенно тогда, когда решений очень много и для каждого из них пришлось бы выполнять команду `eval()`, если использовать предыдущий подход. Для решения нашей системы вызов команды `map()` выглядит так:

```
> map(subs,[sols],eqns);
```

$$[\{1 = 1, 3 = 3\}, \{1 = 1, 3 = 3\}]$$

Команда `solve()` может решать неопределенные системы уравнений, в которых количество уравнений меньше числа неизвестных. В этом случае система Maple сама решает, какие из неизвестных принять за параметры, а какие за неизвестные, относительно которых следует строить решение:

```
> eqn1:=x+2*y+3*z+4*t=41:
> eqn2:=5*x+5*y+4*z+3*t=20:
> sols:=solve({eqn1,eqn2});
```

$$sols := \{y = 37 - \frac{11}{5}z - \frac{17}{5}t, x = -33 + \frac{7}{5}z + \frac{14}{5}t, z = z, t = t\}$$

Здесь решение получено в параметрической форме относительно неизвестных z и t, которые выбраны системой. Можно явно указать, относительно каких неизвестных следует решать систему уравнений, тогда оставшиеся будут рассматриваться как параметры:

```
> sols1:=solve({eqn1,eqn2},{y,z});
```

$$sols1 := \{z = \frac{5}{7}x + \frac{165}{7} - 2\,t, y = -\frac{11}{7}x - \frac{104}{7} + t\}$$

В этом решении явно указаны неизвестные y и z, и полученное решение зависит от двух параметров x и t.

С помощью функции `eval()` можно вычислить значения решения при конкретных значениях параметров и даже выделить их в отдельные переменные:

```
> eval(sols1,{x=1,t=1});
```

$$\{z = \frac{156}{7}, y = \frac{-108}{7}\}$$

```
> x1:=eval(x,sols1);
```

$$x1 := -33 + \frac{7}{5} z + \frac{14}{5} t$$

```
> y1(1,1);
```

$$-\frac{11}{7} \mathrm{x}(1, 1) - \frac{104}{7} + \mathrm{t}(1, 1)$$

Однако выделенное решение не совсем удобно для дальнейшего применения: оно не является функцией своих параметров и каждый раз, когда необходимо вычислить его при каких-то значениях параметров, необходимо обращаться к функции `eval()`. Хотелось бы получить его в виде функции от двух переменных. Для этого следует воспользоваться командой `unapply()`, которая преобразует выражение в функцию. Первым параметром задается само выражение, а последующие определяют, от каких переменных эта функция будет зависеть:

```
> y1:=unapply(y1,x,t);
```

$$y1 := (x, t) \to -\frac{11}{7} x - \frac{104}{7} + t$$

```
> y1(1,1);
```

$$\frac{-108}{7}$$

```
> z1:=unapply(eval(z,sols1),x,t);
```

$$z1 := (x, t) \to \frac{5}{7} x + \frac{165}{7} - 2\,t$$

```
> z1(1,1);
```

$$\frac{156}{7}$$

Если при решении систем уравнений ответ получается в виде множества уравнений, в которых левая часть является неизвестной переменной, то для того, чтобы присвоить найденные значения переменным, относительно которых решалась система, следует применять команду `assign()`. Эта команда присваивает переменным, стоящим в левой части уравнений из множества решений, значения, равные правым частям. Об этой команде можно мыслить как о команде, которая заменяет знак равенства (=) на знак операции

присваивания (:=) во множестве, состоящем из уравнений, в которых левые части представлены неизвестными:

```
> {q=a+b,w=g+p};
```

$$\{q = a + b, w = g + p\}$$

```
> assign(%);q;w;
```

$$a + b$$

$$g + p$$

```
> eq:=x*a+y*b=c;
```

$$eq := x\,a + y\,b = c$$

```
> s:=solve({eq,x+y=1},{x,y});
```

$$s := \{y = \frac{a - c}{a - b}, x = -\frac{b - c}{a - b}\}$$

```
> assign(s); x; y;
```

$$-\frac{b - c}{a - b}$$

$$\frac{a - c}{a - b}$$

Если решение получено в виде последовательности выражений, то получить значение соответствующего решения можно с помощью индекса.

```
> eq:=x^4+2*x^2+1=0;
```

$$eq := x^4 + 2\,x^2 + 1 = 0$$

```
> s:=solve(eq);
```

$$s := I, -I, I, -I$$

```
> x1:=s[2]; x1;
```

$$x1 := -I$$

$$-I$$

Напомним, что в приведенном примере I означает комплексную мнимую единицу, равную $\sqrt{-1}$.

2.4.2. Команда *fsolve()*

По умолчанию Maple пытается найти аналитическое выражение для корней уравнения. Если это ему не удается, то, как отмечалось выше, он просто ничего не печатает в области вывода. В подобных случаях (если корни действительно существуют) можно воспользоваться командой `fsolve()`, которая находит численное решение уравнения или системы уравнений. Формат команды отличается от формата команды `solve()` наличием третьего параметра опция:

```
fsolve(уравнения, переменные, опция);
```

Задание первых двух параметров соответствует заданию аналогичных параметров в команде `solve()`, а параметр `опция` может принимать значения из табл. 2.7.

Таблица 2.7. *Значения параметра* `опция` *команды* `fsolve()`

Значение	Смысл
`complex`	Разыскиваются комплексные корни (только для полиномов)
`Fulldigits`	Используется арифметика с максимальной мантиссой
`Maxsols=n`	Разыскивается `n` решений (только для полиномов)
`a..b` или `x=a..b`	Задан промежуток $[a, b]$, на котором разыскивается решение (во второй форме задания этой опции `x` обозначает имя неизвестной переменной в уравнении)

По умолчанию для произвольного уравнения эта функция находит одно решение, но для полиномов определяются все действительные корни. Чтобы найти все корни полинома, включая комплексные, следует задать опцию `complex`. Использование команды численного решения уравнений показано в примере 2.51.

Пример 2.51. Численное решение уравнений

```
> eq:=x^4+2*x^2-1=0;
```

$$eq := x^4 + 2\,x^2 - 1 = 0$$

```
> s:=fsolve(eq,x);
```

$$s := -.6435942529, .6435942529$$

```
> s:=fsolve(eq,x,complex);
```

$$s := -.6435942529, -1.553773974\ I, 1.553773974\ I, .6435942529$$

```
> fsolve(ln(sin(x))=0,x);
```

$$1.570796327$$

```
> fsolve(ln(sin(x))=0,x,x=2..infinity);
```

$$14.13716694$$

```
> fsolve(ln(sin(x))=0,x,x=15..infinity);
```

$$20.42035225$$

В этом примере также показано, каким образом можно последовательно находить корни произвольного уравнения, задавая интервал изменения неизвестной величины с учетом полученного решения на предыдущем шаге нахождения корня (последние три команды).

2.4.3. Другие команды решения уравнений

Кроме универсальных команд solve() и fsolve() решения уравнений и систем уравнений, Maple содержит специализированные команды, предназначенные для решения либо определенного класса уравнений, либо нахождение решений в заданном числовом поле. В данном разделе эти команды описаны очень кратко, исключительно для того, чтобы читатель знал об их существовании. Более подробную информацию всегда можно получить в справочной системе Maple, выполнив команду ?имя_команды, где вместо параметра имя_команды следует подставить ее действительное имя.

Команда isolve() ищет все целые решения уравнений. Если в уравнении задано несколько неизвестных, то строится решение относительно всех заданных неизвестных.

Пример 2.52. Целочисленное решение уравнений

```
> isolve({(x-1)*(x-1/2)=0});
```

$$\{x=1\}$$

```
> isolve({3*x+4*y=1});
```

$$\{x=-1-4_Z1, y=1+3_Z1\}$$

В решении последнего уравнения примера 2.52 использована целочисленная переменная $_Z1$, сгенерированная Maple.

Команда msolve() также ищет целочисленные решения уравнения, но только по модулю, заданному вторым параметром.

Пример 2.53. Целочисленное решение уравнений по заданному целому модулю

```
> solve({3*x-4*y=1,7*x+y=2});
```

$$\{x=\frac{9}{31}, y=\frac{-1}{31}\}$$

```
> msolve({3*x-4*y=1,7*x+y=2},11);
```

$$\{x=1, y=6\}$$

```
> msolve({3^n=4},11);
```

$$\{n=4+5_Z1\sim\}$$

Для решения рекуррентных уравнений в Maple включена специальная команда rsolve(), которая строит общее решение рекуррентного уравнения, используя начальные значения, если они заданы, или через их символьные обозначения, если они не заданы.

Пример 2.54. Решение рекуррентных уравнений

```
> rsolve({F(n+2)=F(n+1)+F(n)},F(n)); # Без начальных условий
```

$$\frac{1}{5}\,\frac{\sqrt{5}\,(\mathrm{F}(1)\sqrt{5}-\mathrm{F}(1)-\mathrm{F}(0)\sqrt{5}+3\,\mathrm{F}(0))\left(2\,\frac{1}{\sqrt{5}-1}\right)^{n}}{\sqrt{5}-1} + \frac{\left(-\mathrm{F}(1)-\frac{1}{5}\mathrm{F}(1)\sqrt{5}+\mathrm{F}(0)+\frac{3}{5}\mathrm{F}(0)\sqrt{5}\right)\left(-2\,\frac{1}{\sqrt{5}+1}\right)^{n}}{\sqrt{5}+1}$$

```
> rsolve({F(n+2)=F(n+1)+F(n), F(0)=1, F(1)=1},{F(n)});
                     # Используя заданные начальные условия
```

$$\left\{\mathrm{F}(n)=-\frac{2}{5}\,\frac{\sqrt{5}\left(-2\,\frac{1}{-\sqrt{5}+1}\right)^{n}}{-\sqrt{5}+1}+\frac{\frac{2}{5}\sqrt{5}\left(-2\,\frac{1}{\sqrt{5}+1}\right)^{n}}{\sqrt{5}+1}\right\}$$

2.4.4. Решение неравенств

Для решения неравенств и систем неравенств в области вещественных чисел следует использовать команду `solve()` точно так же, как и для решения уравнений и систем уравнений. Ответ выражается либо в виде множества неравенств, либо через функции `RealRange()` и `Open()`. Первая определяет замкнутый отрезок действительных чисел, а вторая используется для указания того, что граничная точка не входит в построенное решение. Для того чтобы задать решение в виде множества, следует задать в виде множества либо само неравенство, либо неизвестную, относительно которой ищется решение. Если этого не сделать, то ответ будет получен с использованием указанных функций определения действительных отрезков.

Пример 2.55. Решение неравенств

```
> solve( (x+2)/(3-x)>2,x);
```

$$\mathrm{RealRange}\left(\mathrm{Open}\left(\frac{4}{3}\right),\mathrm{Open}(3)\right)$$

```
> solve( (x+2)/(3-x)>2,{x});
```

$$\left\{\frac{4}{3}<x, x<3\right\}$$

```
> solve(log[1/3](log[2](x^2-8))>=-1);
```

$$\mathbf{RealRange\ (Open\,(-4\,),\ Open\,(-3\,)),\ RealRange\ (Open\,(3\,),\ Open\,(4\,))}$$

```
> solve({log[1/3](log[2](x^2-8))>=-1});
```

$$\{-4<x, x<-3\},\ \{3<x, x<4\}$$

В примере 2.55 решены два неравенства, для каждого из которых построено решение в виде множества и в форме действительных интервалов. Пользователь сам решает, какая форма ответа ему более подходит.

2.5. Дифференцирование и интегрирование

Maple позволяет вычислять обыкновенные и частные производные аналитического выражения по одной или нескольким переменным. Для этой процедуры предназначены команды `diff()` и `Diff()`. Вторая команда является так называемой *отложенной* командой (inert command), которая не вычисляет производную от выражения, а просто отображает математическую запись взятия производной. Результат действия отложенной команды можно присвоить переменной Maple, а в дальнейшем при помощи команды `value()` вычислить результат этой отложенной команды. Отложенная форма команды дифференцирования удобна, когда необходимо видеть, какие операции были сделаны для получения нужного выражения. Кроме отложенной команды дифференцирования в Maple есть еще целый ряд команд, имеющих отложенную форму, полную информацию о которых можно получить в Справке.

Синтаксис команды дифференцирования следующий:

```
diff(выражение, переменная_1, переменная_2, ..., переменная_n);
diff(выражение, [переменная_1, переменная_2, ..., переменная_n]);
```

В результате выполнения любой из приведенных команд будет вычислена частная производная *n*-го порядка от заданного первым параметром выражения по заданным *n* переменным.

При формировании производных высокого порядка полезен оператор последовательности `$`, который позволяет проще и нагляднее задать производную. Например, для вычисления третьей производной функции `f(x)` по переменной *x* можно использовать команду `diff(f(x),x,x,x)`, в которой три раза указано дифференцирование по переменной *x*, или применить в команде дифференцирования оператор последовательности `x$3`, что упрощает и делает более наглядным задание третьей производной: `diff(f(x),x$3)`.

Пример 2.56. Вычисление производных

```
> f:=x^2*sin(x)+sqrt(y)*ln(cos(x));
```

$$f := x^2 \sin(x) + \sqrt{y}\ \ln(\cos(x))$$

```
> diff(f,x);
```

$$2\,x \sin(x) + x^2 \cos(x) - \frac{\sqrt{y}\,\sin(x)}{\cos(x)}$$

```
> diff(f,x$2);
```

$$2\sin(x)+4\,x\cos(x)-x^2\sin(x)-\sqrt{y}-\frac{\sqrt{y}\,\sin(x)^2}{\cos(x)^2}$$

```
> diff(f,x,y);
```

$$-\frac{1}{2}\,\frac{\sin(x)}{\sqrt{y}\,\cos(x)}$$

```
> fDerive:=Diff(f,x);
```

$$fDerive := \frac{\partial}{\partial x}(x^2\sin(x)+\sqrt{y}\,\ln(\cos(x)))$$

```
> g:=sqrt(fDerive);
```

$$g := \sqrt{\frac{\partial}{\partial x}(x^2\sin(x)+\sqrt{y}\,\ln(\cos(x)))}$$

```
> value(%);
```

$$\sqrt{2\,x\sin(x)+x^2\cos(x)-\frac{\sqrt{y}\,\sin(x)}{\cos(x)}}$$

Последние три команды показывают использование отложенной формы команды дифференцирования.

Интегрирование выражений по заданной переменной осуществляется командой `int()`, которая также имеет отложенную форму `Int()`. Эта команда позволяет вычислять как неопределенный интеграл от выражения (при этом, правда, в ответе не будет никакой постоянной интегрирования) с использованием следующего синтаксиса команды:

```
int( выражение, переменная);
```

так и определенный интеграл с помощью следующего синтаксиса команды

```
int( выражение, переменная=a..b);
```

где `a` и `b` являются пределами интегрирования, причем эти пределы могут быть и аналитическими выражениями.

Пример 2.57. Интегрирование функций

```
> f:=a*x^2*sin(b*x);
```

$$f := a\,x^2\sin(b\,x)$$

```
> int(f,x);
```

$$\frac{a\,(-b^2x^2\cos(b\,x)+2\cos(b\,x)+2\,b\,x\sin(b\,x))}{b^3}$$

```
> int(f,x=0..1);
```

$$-\frac{a\,(b^2\cos(b)-2\cos(b)-2\,b\sin(b)+2)}{b^3}$$

```
> int(f,x=0..a);
```

$$-\frac{a\,(b^2\cos(b\,a)\,a^2-2\cos(b\,a)-2\,b\sin(b\,a)\,a+2)}{b^3}$$

```
> Int(f,x=0..Pi);
```

$$\int_0^{\pi} a\,x^2\sin(b\,x)\,dx$$

```
> value(%);
```

$$-\frac{a\,(-2\cos(\pi\,b)+b^2\cos(\pi\,b)\,\pi^2-2\,b\sin(\pi\,b)\,\pi+2)}{b^3}$$

Для символьного вычисления определенного интеграла существуют две опции, управляющие обработкой разрывов подынтегральной функции. Эти опции задаются третьим параметром в командах `int()` и `Int()`.

По умолчанию команда интегрирования проверяет выражение на непрерывность в области интегрирования и вычисляет интеграл как сумму отдельных определенных интегралов на промежутках непрерывности функции. Опция `` `continuous` `` отключает этот режим и вычисляет интеграл как разность значений первообразной подынтегральной функции в точке начала и конца промежутка интегрирования.

Еще одна опция `` `CauchyPrincipalValue` `` вычисляет несобственные интегралы первого и второго рода в смысле главного значения Коши.

Если Maple не может найти замкнутую форму выражения для определенного интеграла, то команда интегрирования возвращает просто вызов самой себя (в области вывода печатается математическая запись вычисления интеграла, как при обращении к отложенной команде интегрирования). В подобных случаях можно вычислить значение определенного интеграла численным способом с помощью команды `evalf()`. Синтаксис подобной конструкции следующий:

```
evalf( int(f,x=a..b));
evalf( Int(f,x=a..b));
evalf( Int(f,x=a..b), digits, flag);
```

Параметр `digits` позволяет задать число значащих цифр при вычислениях приближенного значения интеграла (по умолчанию это число равно числу значащих цифр, определенных значением системной константы `Digits`).

При численном интегрировании по умолчанию используется квадратурная формула Кленшо-Куртиса (Clenshaw-Curtis). Если в подынтегральном выражении встречается сингулярность, то применяется специальная методика символьного анализа для ее разрешения. Для задач с неустранимыми сингулярностями используется адаптивный метод двойных экспоненциальных квадратур. Параметр `flag` позволяет явно задать метод численного интегрирования. Он может принимать значения, представленные в табл. 2.8.

Таблица 2.8. *Значения параметра* `flag` *при численном интегрировании*

Значение	Смысл
`_Ccquad`	Применяется только квадратура Кленшо-Куртиса без вызова процедуры обработки сингулярности
`_Dexp`	Применяется адаптивный метод двойных экспоненциальных квадратур
`_Ncrule`	Применяется метод квадратурной формулы Ньютона-Котеса, являющийся методом фиксированного порядка, и не эффективен для высоких точностей (`Digits>15`)

Несколько примеров численного вычисления интегралов помогут освоиться с использованием вышеприведенной методики.

Пример 2.58. Численное интегрирование функций

```
> int(sin(x)*ln(x),x=0..1);
```

$$\mathrm{Ci}(1)-\gamma$$

```
> evalf( int(sin(x)*ln(x),x=0..1));
```

$$-.2398117420$$

```
> Int(sin(x)*ln(x),x=0..1)=evalf(Int(sin(x)*ln(x),
                          x=0..1,20,_Dexp));
```

$$\int_0^1 \sin(x)\ln(x)\,dx = -.23981174200056472594$$

```
> Int(1/(1+x^2),x=0..infinity)=evalf(Int(1/(1+x^2),
                          x=0..infinity,30,_Dexp));
```

$$\int_0^\infty \frac{1}{1+x^2}\,dx = 1.57079632679489661923132169164$$

```
> Int(exp(x-x^2/2)/(1+exp(x)/2),x=-infinity..infinity)=
    evalf(Int(exp(x-x^2/2)/(1+exp(x)/2),x=-infinity..infinity));
```

$$\int_{-\infty}^{\infty} \frac{\mathrm{e}^{(x-1/2\,x^2)}}{1+\frac{1}{2}\mathrm{e}^x}\,dx = 1.805577062$$

Первый интеграл примера 2.58 вычисляется в аналитическом виде, но представляется через значение специальной функции интегральный косинус. Для получения ответа в виде десятичного числа применяется алгоритм численного интегрирования. На этом же примере показано использование от-

ложенной формы команды интегрирования для более удобного представления ответа.

Замечание

Численное интегрирование даже функций, внешний вид которых кажется не достаточно сложным, может потребовать значительного времени. Если будет казаться, что Maple завис, а такое случается, следите за изменением времени в правой части строки состояния. Если оно изменяется, то просто следует дождаться завершения интегрирования.

В заключение этого раздела отметим, что в системе Maple имеется набор команд *для полного исследования* функций: `limit()` — *для* отыскания предела функции, `sum()` — для нахождения всевозможных конечных сумм, `series()` — для разложения функций в ряды Тейлора, Маклорена и Лорана, `extrema()` — для исследования экстремумов функций как одной, так многих переменных, `minimize()` и `maximize()` — для поиска минимума и максимума функции на заданном промежутке. Описание всех этих и других команд можно, естественно, найти в Справке Maple. В частях II и III нашей книги мы познакомим читателя с большинством перечисленных команд.

2.6. Решение обыкновенных дифференциальных уравнений

Для решения обыкновенных дифференциальных уравнений и их систем можно воспользоваться командой `dsolve()` или функциями пакета `DEtools`, которые позволяют строить численное решение дифференциальных уравнений и отображать их в графическом виде, включая построение фазового портрета систем дифференциальных уравнений (описание этого пакета можно найти в гл. 3). В данном разделе мы остановимся только на описании команды `dsolve()`, с помощью которой можно не только получить аналитическое решение дифференциального уравнения, но и сформировать процедуру построения численного решения задачи Коши, если система Maple не сможет найти общее решение в аналитическом виде.

Наиболее общий синтаксис вызова команды решения дифференциального уравнения следующий:

```
dsolve(уравнения, неизвестные, [опции]);
```

Параметром `уравнения` задается одно дифференциальное уравнение или система дифференциальных уравнений. В последнем случае все уравнения системы должны быть представлены в виде множества (их список через запятую следует заключить в фигурные скобки). Параметр `неизвестные` определяет неизвестную функцию дифференциального уравнения или неизвестные функции системы дифференциальных уравнений, которые, как и сами

уравнения системы, должны быть представлены в виде множества. Необязательный параметр опции, определяемый в виде ключевое_значение=значение, позволяет задать методы и форму представления решения.

Для задания производной искомой функции в дифференциальном уравнении можно использовать команду diff() или оператор D, причем саму неизвестную функцию следует определять с явным указанием независимой переменной, например y(x). С командой diff() мы уже знакомы *(см. раздел 2.5 "Дифференцирование и интегрирование")*, а вот оператор D встретился нам впервые. Он определяет операцию дифференцирования и имеет следующий синтаксис:

```
(D@@n)(функция)(переменная);
```

В этой записи n представляет целое число, определяющее порядок производной, параметр функция — используемый идентификатор функции, а параметр переменная — независимую переменную функции. Например, производную второго порядка функции f(x) с использованием этого оператора следует задавать следующим образом:

```
(D@@2)(f)(x);
```

Несколько примеров задания дифференциальных уравнений и систем дифференциальных уравнений представлены ниже:

```
> eqn1:=diff(y(x),x$2)+k^2*y(x)=0;
```

$$eqn1 := \left(\frac{\partial^2}{\partial x^2}\,\mathrm{y}(x)\right) + k^2\,\mathrm{y}(x) = 0$$

```
> eqn2:=(D@@2)(y)(x)+k^2*y(x)=sin(k1*x);
```

$$eqn2 := (\mathrm{D}^{(2)})(y)(x) + k^2\,\mathrm{y}(x) = \sin(k1\ x)$$

```
> sys1:={D(y1)(x)=a[1,1]*y1(x)+a[1,2]*y2(x),
        D(y2)(x)=a[2,1]*y1(x)+a[2,2]*y2(x)};
```

$$sys1 := \{\mathrm{D}(y2)(x) = a_{2,1}\,\mathrm{y1}(x) + a_{2,2}\,\mathrm{y2}(x), \mathrm{D}(y1)(x) = a_{1,1}\,\mathrm{y1}(x) + a_{1,2}\,\mathrm{y2}(x)\}$$

Обратите внимание, что в приведенных примерах и уравнения, и система уравнений сохраняются в переменных Maple. Как уже отмечалось ранее, это достаточно распространенный прием, позволяющий использовать в дальнейшем заданные уравнения простой ссылкой на обычную переменную.

Попробуем решить первое уравнение из приведенных примеров. Для этого достаточно выполнить следующую команду dsolve():

```
> dsolve(eqn1,y(x));
```

$$\mathrm{y}(x) = _C1\,\cos(k\,x) + _C2\,\sin(k\,x)$$

Найдено общее решение дифференциального уравнения, в котором переменные *_C1* и *_C2* — это сгенерированные Maple специальные переменные, представляющие произвольные константы общего решения дифферен-

циального уравнения второго порядка. Этот пример показывает, что при отсутствии каких-либо опций Maple пытается найти точное общее решение в явном виде. Если в явном виде решения не существует, то система попытается найти его в неявном виде, как видно из следующего примера:

```
> eq:=diff(y(x),x)=sqrt(x^2-y(x))+2*x;
```

$$eq := \frac{\partial}{\partial x}\mathrm{y}(x) = \sqrt{x^2-\mathrm{y}(x)} + 2\,x$$

```
> dsolve(eq,y(x));
```

$$4\,\frac{\mathrm{y}(x)\,x}{2\,\sqrt{x^2-\mathrm{y}(x)}-x} + \frac{8\,\mathrm{y}(x)\,\sqrt{x^2-\mathrm{y}(x)}}{2\,\sqrt{x^2-\mathrm{y}(x)}-x} - \frac{3\,x^3}{2\,\sqrt{x^2-\mathrm{y}(x)}-x} - \frac{6\,x^2\,\sqrt{x^2-\mathrm{y}(x)}}{2\,\sqrt{x^2-\mathrm{y}(x)}-x} - _C1 = 0$$

```
> isolate(%,y(x));
```

$$\mathrm{y}(x) = \frac{5}{4}x^2 + \frac{1}{2}(-x-\sqrt{-_C1}\,)\,x + \frac{1}{4}_C1$$

Команда `isolate()` в этом примере выражает заданное вторым параметром выражение (`y(x)`) из уравнения, определяемого первым параметром (в нашем случае из неявного вида общего решения дифференциального уравнения).

По умолчанию команда `dsolve()` сначала пытается найти общее решение в явном виде, и если таковое не удается найти, то решение выдается в неявном виде (конечно, при условии его существования). Можно "озадачить" Maple поиском общего решения в явном виде, используя опцию `explicit=true` (по умолчанию используется `explicit=false`):

```
> dsolve(eq,y(x),explicit=true);
```

$$\mathrm{y}(x) = \mathrm{RootOf}(4\,_Z\,x + 8\,_Z\sqrt{x^2-_Z} - 3\,x^3 - 6\,x^2\sqrt{x^2-_Z} - 2\,_C1\sqrt{x^2-_Z} + x\,_C1)$$

Как видим, в этом случае мы действительно получили сразу же решение в явном виде, но оно представлено через функцию `RootOf()`, так что наш первоначальный подход к решению дифференциального уравнения оказался более продуктивным.

Найти общее решение в явном или неявном виде удается не для любого дифференциального уравнения. В этом случае можно построить приближенное решение в форме ряда Тейлора. Для этого следует задать опцию `type=series` в команде `dsolve()` (по умолчанию используется `type=exact`), а также установкой значения системной переменной `Order` определить, до какого порядка малости относительно независимой переменной функции ищется разложение решения в ряд Тейлора в окрестности нулевой точки:

```
> Order:=3;
> eq:=(D@@2)(y)(x)+(a+b*x^2)*D(y)(x)+y(x)=0;
```

$$eq := (\mathrm{D}^{(2)})(y)(x) + (a+b\,x^2)\,\mathrm{D}(y)(x) + \mathrm{y}(x) = 0$$

```
> dsolve(eq,y(x),type=series);
```

$$y(x) = y(0) + D(y)(0)\,x + \left(-\frac{1}{2}D(y)(0)\,a - \frac{1}{2}y(0)\right)x^2 + O(x^3)$$

Обратите внимание, что в решении дифференциального уравнения второго порядка, представленном рядом Тейлора, в качестве двух произвольных постоянных используются значения искомой функции и ее первой производной в точке `x=0`: $y(0)$, $D(y)(0)$.

Решение задачи Коши или краевой задачи для дифференциального уравнения в системе Maple так же просто, как и отыскание общего или приближенного решения. Для этого необходимо задать первый параметр команды `dsolve()` в виде множества, элементами которого являются само уравнение и все начальные или краевые условия. Решим задачу Коши и краевую задачу для следующего дифференциального уравнения второго порядка:

```
> eqn1:=diff(y(x),x$2)+k^2*y(x)=0;
```

$$eqn1 := \left(\frac{\partial^2}{\partial x^2}y(x)\right) + k^2\,y(x) = 0$$

Задача Коши для этого дифференциального уравнения второго порядка требует задания в нулевой точке значения неизвестной функции и ее первой производной. Ее решение представлено ниже:

```
> dsolve({eqn1,y(0)=0,D(y)(0)=1},y(x));
```

$$y(x) = \frac{\sin(k\,x)}{k}$$

Краевая задача для этого дифференциального уравнения второго порядка требует задания в двух точках, например, `x=0` и `x=1` значения неизвестной функции. Ее решение также получено с помощью команды `dsolve()`:

```
> dsolve({eqn1,y(0)=0,y(1)=1},y(x));
```

$$y(x) = \frac{\sin(k\,x)}{\sin(k)}$$

Начальные или краевые условия задаются в виде уравнений, в левой части которых определен задаваемый параметр (значение неизвестной функции или ее производной необходимого порядка) в соответствующей точке, а в правой части значение этого параметра. Обратим внимание, что при задании производных в начальных или краевых условиях следует использовать оператор `D` — команда `diff()` здесь не употребляется.

Если точное решение задачи Коши или краевой задачи система Maple не смогла найти, а приближенное решение в виде ряда Тейлора нас не устраивает, то можно построить численное решение, опять-таки с использованием все той же команды `dsolve()`. Для этого следует задать опцию `type=numeric`,

а с помощью опции `method=метод` определить используемый для построения численного решения метод. Параметр `метод` может принимать одно из значений, представленных в табл. 2.9, в которой также дано краткое описание соответствующего значению метода.

Таблица 2.9. *Значения опции* `method` *при численном решении дифференциальных уравнений*

Значение	Описание
`Rkf45`	Метод Рунге-Кутта-Фальберга порядка 4—5
`Dverk78`	Метод Рунге-Кутта порядка 7—8

По умолчанию (если не задана опция `method`) применяется метод Рунге-Кутта-Фальберга порядка 4—5. Однако при использовании численного решения следует помнить, что все параметры дифференциального уравнения (символьные константы) должны быть определены. Например, для задачи Коши уравнения `eqn1` предыдущего примера следует задать численное значение для параметра `k`.

Численное решение строится в форме процедуры Maple, поэтому следует некоторой переменной присвоить результат построения командой `dsolve()` численного решения в виде процедуры. В дальнейшем имя этой переменной можно использовать как имя процедуры для вычисления значения решения задачи Коши в некоторой точке, соответствующей значению независимой переменной функции решения. Это значение передается в процедуру как ее параметр — после имени процедуры в круглых скобках. Следующий пример демонстрирует построение численного решения задачи Коши и его использование.

```
> eqn1:=diff(y(x),x$2)+k^2*y(x)=0;
```

$$eqn1 := \left(\frac{\partial^2}{\partial x^2}\,y(x)\right) + k^2\,y(x) = 0$$

Переменной `F` присваиваем результат численного решения задачи Коши для дифференциального уравнения второго порядка (в нулевой точке задается значение неизвестной функции и ее первой производной):

```
> F:=dsolve({eqn1,y(0)=0,D(y)(0)=1},y(x),type=numeric);
```

$$F := \mathbf{proc}\,(rkf45_x)\ \ldots\mathbf{end\ proc}$$

(Подробно процедура, ее создание и работа с ней описаны в гл. 5.)

Если не присвоить параметру `k` конкретного числового значения, то попытка получить значение решения в точке, например `x=1`, приведет к ошибке:

```
>F(1);
Error, (in dsolve/numeric/rkf45) cannot evaluate boolean:
2.+abs(.2511886433e-1-.2016799760e-5*k^2-.3377712687e-
4*k^2*(.2318664400e-1-.3700729218e-5*k^2)+.6309573448e-
5*k^2*(.2511886433e-1-.6603721651e-5*k^2)) <= 0.
```

Следует обязательно определить все символьные параметры дифференциального уравнения числовыми значениями перед использованием численного решения:

```
> k:=1:
> F(0);F(1);F(2);
```

$$\left[x = 0, y(x) = 0., \frac{\partial}{\partial x} y(x) = 1.\right]$$

$$\left[x = 1, y(x) = .841470989048380025, \frac{\partial}{\partial x} y(x) = .540302309040994189\right]$$

$$\left[x = 2, y(x) = .909297437216234794, \frac{\partial}{\partial x} y(x) = -.416146840428394782\right]$$

Обратите внимание, в каком виде построенная процедура численного решения выдает результаты — в виде списка значений независимой переменной, самой функции и ее производных (до порядка на единицу меньше порядка самого уравнения).

ГЛАВА 3



Пакеты

Maple, как и любая другая программная система, имеет собственную архитектуру. Ядро, основная библиотека и пакеты — все это элементы, составляющие в совокупности Maple. Пользователь, работая в системе аналитических вычислений Maple, постоянно "общается" с этими тремя компонентами: одни команды находятся в постоянно находящемся в памяти компьютера ядре, другие автоматически загружаются из основной библиотеки, а для выполнения третьих следует явным образом подключить соответствующий пакет. Все это в совокупности и составляет предмет организации Maple.

3.1. Организация Maple

При запуске Maple в память компьютера загружается только ядро, являющееся его основным компонентом. В нем содержатся программы, без которых Maple не может функционировать, а также программы, реализующие основные низкоуровневые команды выполнения простых аналитических преобразований и используемые командами из основной библиотеки и пакетов. К ним относятся интерпретатор языка Maple, программы работы с числовыми данными, а также программы, отображающие результаты выполнения команд Maple и выполняющие другие операции ввода/вывода. Ядро состоит из программ, написанных на языке С, и составляет около 10% общего размера системы аналитических вычислений.

Остальная часть программ, реализующих функциональность Maple, написана на языке Maple и хранится в библиотеке, которая состоит из двух частей: основной библиотеки и многочисленных пакетов, в которых сгруппированы команды для выполнения определенных математических действий.

Основная библиотека содержит наиболее часто используемые команды Maple, которые загружаются автоматически, как только пользователь обра-

щается к ним. В отличие от предыдущих версий не стоит заботиться об их явной загрузке. Программы на языке Maple представляют собой очень компактные процедуры, быстро загружающиеся в память и выполняющиеся с достаточно высокой скоростью, так что пользователь практически не может отличить, какая команда находится в ядре, а какая загружается из библиотеки.

Если необходимая для выполнения математических преобразований команда не находится в ядре или основной библиотеке, то пользователю приходится явно подключать пакет или только одну программу из пакета, чтобы иметь возможность обратиться к требуемой команде.

Пакеты в Maple используются для удобства организации работы пользователя. Пакет представляет собой набор команд для решения задач, относящихся к определенным разделам математики, или решения определенных задач графического представления информации, например, пакет `finance` служит для решения задач финансовой математики, в пакете `stats` собраны команды для статистической обработки результатов и т. д.

Для того чтобы использовать команды какого-нибудь пакета, необходимо подключить его, так как они все находятся не в ядре системы Maple, а в специальных файлах. Подключение пакетов осуществляется с помощью команды

```
with(пакет);
```

где в качестве параметра указывается имя соответствующего пакета. Может оказаться, что подключаемый пакет содержит команду с таким же именем, что и в ранее подключенном пакете. В этом случае в области вывода отображается сообщение о переопределении соответствующей команды, и результат ее действия будет соответствовать команде, находящейся в последнем подключенном пакете. Подключив пакет, в дальнейшем пользователь может вызывать все его команды, просто набирая их имя и требуемые для ее выполнения параметры прямо в области ввода.

Если необходима какая-то конкретная команда пакета, то вместо подключения всего пакета целиком можно подключить эту одну команду с помощью следующего синтаксиса оператора `with()`:

```
with(пакет, имя_команды);
```

В дальнейшем эта команда также может быть использована ссылкой только на ее имя, без указания пакета, в котором она находится.

Если пользователь не желает, чтобы команда постоянно находилась в памяти, можно ее загрузить только на время ее выполнения, после чего она будет выгружена из памяти. Для этого следует указать ее полное имя, состоящее из имени пакета и имени самой команды, в следующем виде:

```
имя_пакета[имя_команды](...);
```

При следующем обращении к команде ее необходимо снова загрузить одним из трех указанных способов. Пример 3.1 демонстрирует подключение пакета и вызов его команды.

Пример 3.1. Подключение пакета

```
> with(combstruct);
```

[*allstructs*, *count*, *draw*, *finished*, *gfeqns*, *gfseries*, *gfsolve*, *iterstructs*, *nextstruct*]

```
> bin := {B=Union(Z,Prod(B,B))}:
> count([B,bin,unlabelled],size=7);
```

132

Если пользователь не желает, чтобы отображались все команды подключаемого пакета (список команд некоторых пакетов может занимать целую страницу), то команду `with()` следует завершить двоеточием (:).

Встроенные в Maple пакеты позволяют выполнять математические построения и преобразования, начиная от элементарной математики и заканчивая общей теорией относительности. Табл. 3.1 содержит список всех пакетов Maple с их кратким описанием и дает представление о возможностях Maple.

Таблица 3.1. *Пакеты Maple*

Наименование пакета	Содержит
`algcurves`	Средства для изучения одномерных алгебраических кривых, определяемых полиномами нескольких переменных
`codegen`	Средства для создания, обработки и перевода процедур Maple в код языков программирования С и Fortran
`combinat`	Комбинаторные функции, включая вычисление перестановок и сочетаний. В настоящее время считается устаревшим и рекомендуется для работы с комбинаторными задачами использовать пакет `combstruct`
`combstruct`	Команды для создания и работы с комбинаторными структурами
`context`	Средства для построения и изменения контекстных меню в графическом интерфейсе пользователя
`DEtools`	Средства для выполнения преобразований обыкновенных дифференциальных уравнений, их решения и графического отображения решений с возможностью построения фазовых портретов и полей направлений систем дифференциальных уравнений
`diffalg`	Команды для работы с системами полиномиальных дифференциальных уравнений, как обыкновенных, так и в частных производных

Таблица 3.1 (продолжение)

Наименование пакета	Содержит
`difforms`	Команды для работы с дифференциальными формами при решении задач дифференциальной геометрии
`Domains`	Команды для создания областей вычисления. Поддерживают работу с полиномами, матрицами и рядами над числовыми кольцами, конечными полями, кольцами полиномов и матриц
`finance`	Команды для выполнения финансовых вычислений (финансовая математика)
`GaussInt`	Команды для работы с гауссовыми целыми числами — комплексными числами вида *a+bI*, где *a* и *b* целые
`genfunc`	Команды для работы с рациональными производящими функциями
`geom3d`	Команды для выполнения построений и вычислений в трехмерном евклидовом пространстве. Позволяют строить и работать в трехмерном пространстве с точками, линиями, плоскостями, треугольниками, сферами и т. д.
`geometry`	Команды для выполнения построений и вычислений на евклидовой плоскости. Позволяют строить и работать с точками, линиями, плоскостями, треугольниками, окружностями и т. д.
`GF`	Команды для работы с полями Галуа
`Groebner`	Команды для организации вычислений в базисе Гребнера
`group`	Команды для работы с группами перестановок и конечными группами
`inttrans`	Команды для работы с интегральными преобразованиями и их обратными преобразованиями
`liesymm`	Команды определения симметричности систем дифференциальных уравнений в частных производных
`linalg`	Команды работы с символьными матрицами и векторами: сложение, умножение матриц, собственные числа и векторы в символьном виде и т. д.
`LinearAlgebra`	Усовершенствованные команды линейной алгебры для работы со специальным типом числовых матриц `Matrix`
`LREtools`	Команды для преобразования, графического отображения и решения линейных рекуррентных уравнений
`Matlab`	Команды для подключения и использования некоторых матричных функций системы численных вычислений Matlab, включая нахождение собственных значений и векторов, определителей и вычисление LU-разложения матриц. Работают при установленном пакете Matlab

Таблица 3.1 (продолжение)

Наименование пакета	Содержит
networks	Команды для создания и работы с различными типами графов
numapprox	Команды для построения полиномиальной аппроксимации функций на заданном интервале
numtheory	Команды для вычислений в области классической теории чисел: является ли число простым, нахождение *n*-го простого числа, разложение чисел на простые множители и т. д.
Ore_algebra	Программы для основных вычислений в алгебрах линейных операторов
orthopoly	Команды построения различных типов ортогональных полиномов
padic	Команды *p*-адического приближения вещественных чисел
PDEtools	Средства для выполнения преобразований дифференциальных уравнений в частных производных, их решения и графического отображения решений
plots	Команды построения специальных видов графиков функций, включая построение линий уровня, отображение неявно заданных функций, включение текстовых надписей в график и построение графиков в различных системах координат
plottools	Команды для создания и работы с графическими объектами
polytools	Команды для работы с полиномами
powseries	Команды построения и работы с формальными степенными рядами
process	Команды, позволяющие писать многопроцессные Maple-программы в системе UNIX
simplex	Команды решения задач линейной оптимизации на основе симплекс-метода
Slode	Команды построения формального решения линейных обыкновенных уравнений в виде степенных рядов
Spread	Команды, позволяющие программировать электронные таблицы Maple
stats	Команды статистической обработки данных
student	Команды выполнения пошаговых вычислений в дифференциальном и интегральном исчислении, включая формулу интегрирования по частям, формулу Симпсона численного приближения определенных интегралов, нахождение максимума и минимума функций

Таблица 3.1 (окончание)

Наименование пакета	Содержит
sumtools	Команды вычисления конечных и бесконечных сумм
tensor	Команды работы с тензорами и их применение в общей теории относительности

В этой главе мы кратко остановимся на некоторых, наиболее полезных, с нашей точки зрения, пакетах, знание работы с которыми позволит быстро решать типовые задачи математического моделирования в различных областях знаний.

Совет

Полный список и описание пакетов можно найти в справочной системе Maple, если выполнить команду `?index[package]`.

3.2. Линейная алгебра

В Maple 6 выполнение преобразований линейной алгебры можно осуществлять с помощью команд двух пакетов: `linalg` и `LinearAlgebra`, функциональность которых практически одинакова. Первый пакет входил в состав и всех предыдущих версий Maple, тогда как второй пакет — это новое средство, позволяющее работать с числовыми матрицами, в том числе и с матрицами больших размеров, используя всю мощь известного пакета численных расчетов NAG (Numerical Algorithms Group).

Основными объектами, с которыми работают команды этих пакетов, являются матрицы, однако матрицы одного пакета не эквивалентны матрицам другого. В пакете `linalg` используются матрицы, построенные на основе массива, создаваемого командой `array()`, тогда как в пакете `LinearAlgebra` применяются векторы и матрицы, построенные на основе новой структуры r-таблицы (r-table) и создаваемые специальными конструкторами `Vector()` и `Matrix()` или с использованием краткой нотации `<a,b,c>`. Матрицы в пакете `linalg` вычисляются только до уровня своих имен, поэтому в нем невозможно вычислить операции поэлементного суммирования или вычитания, используя простые операции над идентификаторами матриц, и приходится пользоваться специальным синтаксисом через команду `evalm()`. В пакете `LinearAlgebra` матрицы вычисляются до уровня своих элементов, поэтому простое задание имени матрицы в области ввода рабочего листа приводит к отображению ее элементов, а не имени матрицы, как в случае с пакетом `linalg`. Кроме этого, в пакете `LinearAlgebra` матрицы могут задаваться в ка-

честве операндов сложения и вычитания, что приводит к поэлементному выполнению указанных операций без использования дополнительных синтаксических конструкций.

Чтобы принять решение, какой пакет линейной алгебры предпочесть, рекомендуется принять во внимание следующие обстоятельства:

- Пакет `linalg` полезен при выполнении абстрактных вычислений над матрицами и векторами.
- Пакет `LinearAlgebra` обладает более дружественным интерфейсом, работает с числовыми матрицами и особенно эффективен при работе с числовыми матрицами больших размеров из-за возможности обращения к откомпилированным программам пакета численных расчетов NAG.

3.2.1. Пакет *linalg*

Пакет линейной алгебры `linalg` содержит команды создания матриц и векторов, предлагает большой набор функций для работы со структурой этих объектов, для выполнения основных матричных и векторных операций и для решения основных задач линейной алгебры: решение систем линейных уравнений, нахождение собственных векторов и собственных векторов матрицы, приведение матриц к специальным формам и т. д. И все эти действия можно выполнять с матрицами и векторами, элементы которых могут быть общими алгебраическими выражениями, получая результаты также в виде алгебраических выражений.

Все команды пакета линейной алгебры работают с матрицами и векторами. В Maple матрицей считается двумерный массив, индексы которого изменяются от единицы. Аналогично, вектор — это одномерный массив с изменяющимся от единицы индексом. Определить матрицу или вектор в Maple можно двумя способами: либо с помощью команды `array()` стандартной библиотеки, либо командами `matrix()` и `vector()` пакета `linalg`.

Наиболее общий синтаксис команды `array()`, которая позволяет задавать многомерные массивы с индексами, изменяющимися в диапазонах целых (как положительных, так отрицательных) чисел, следующий:

```
array(диапазоны,список,опции);
```

Все параметры необязательны и могут задаваться в произвольном порядке. Параметр `диапазоны` представляет собой целочисленные диапазоны изменения индексов массивов, задаваемых через запятую, — размерность массива равна количеству заданных диапазонов. Значения элементов массива задаются параметром `список` в виде списка для одномерных массивов или списка списков для многомерных массивов. В качестве значений параметра `опции` можно применять `symmetric`, `antisymmetric`, `identity` и `diagonal`. Они используются для задания массивов специального вида (симметричных, ан-

тисимметричных, единичных и диагональных). Для задания векторов и матриц с помощью этой функции следует указывать диапазоны изменения индексов, начинающиеся с единицы:

```
> vec:=array(1..2,[1,2]);
```

$$vec := [1,2]$$

```
> matr:=array(1..2,1..2,[[1,2],[10,15]]);
```

$$matr := \begin{bmatrix} 1 & 2 \\ 10 & 15 \end{bmatrix}$$

Для задания тех же вектора и матрицы можно использовать соответственно команды `vector()` и `matrix()` из пакета `linalg`, предварительно подключив его командой `with(linalg)`. Синтаксис этих команд следующий:

```
vector(n,[элемент1,элемент2,...]);
matrix(n,m,[элемент1,элемент2,...]);
```

Здесь целые величины `n` и `m` задают размерности вектора и матрицы, а значения их элементов задаются в виде простого списка.

```
> vec:=vector(2,[1,2]);
```

$$vec := [1,2]$$

```
> matr:=matrix(2,2,[1,2,10,15]);
```

$$matr := \begin{bmatrix} 1 & 2 \\ 10 & 15 \end{bmatrix}$$

Внимание!

В Maple 6 команды `vector()` и `matrix()` находятся в основной библиотеке, поэтому ими можно пользоваться и без подключения пакета `linalg`. Однако доступ ко всем остальным командам этого пакета требует его явного подключения.

Значения элементов вектора или матрицы не обязательно задавать при создании этих объектов. Можно позднее с помощью общепринятой индексной ссылки на элементы вектора или матрицы (в квадратных скобках после имени вектора или массива задаются индекс(ы) требуемого элемента) присвоить им новые значения или использовать уже присвоенные значения в вычислениях.

```
> vec[1]:=5;
```

$$vec_1 := 5$$

```
> eval(vec);
```

$$[5,2]$$

Обратите внимание, что для отображения содержимого вектора `vec` использована команда `eval()`, так как переменная, содержащая сложные объекты, каковыми являются и векторы, и матрицы, вычисляется не полностью, а

только до своего имени. Эту же команду следует применять и в случае, если необходимо посмотреть содержимое матрицы.

Для вычисления некоторых характеристик матриц нужно осуществлять преобразование самих матриц, например, прибавление к одной строке матрицы линейную комбинацию некоторых других, или выделять некоторые подматрицы, например, при вычислении миноров матрицы. Для этих задач можно использовать ряд команд, содержащихся в пакете `linalg`.

Выяснить размерности (количество строк и столбцов) матрицы помогут соответственно команды `rowdim()` и `coldim()`, а определить количество элементов вектора можно командой `vectdim()`. Единственным передаваемым в эти команды параметром является идентификатор матрицы или вектора. Следующий пример демонстрирует использование этих команд:

```
> vectdim(vec);
```

$$2$$

```
> coldim(matr);
```

$$2$$

Иногда необходимо удалить или, наоборот, добавить в матрицу несколько строк или столбцов. Удалить строки или столбцы с номерами от `i` до `j` из матрицы `A` можно соответственно командами `delrows(A,i..j)` и `delcols(A,i..j)`. Для удаления одной строки или одного столбца следует использовать диапазон `i..i`. Добавить строки и столбцы в матрицу `A` можно командой `extend(A,rows,cols,expr)`, в которой параметры `rows` и `cols` являются целыми числами, включая `0`, и представляют соответственно количество добавляемых строк и столбцов. Параметр `expr` — выражение, значение которого используется в качестве значений добавляемых элементов строк и столбцов.

```
> A:=matrix(2,2,1);
```

$$A := \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

```
> delcols(A,1..1);
```

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

```
> A1:=extend(A,1,0,-1);
```

$$A1 := \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ -1 & -1 \end{bmatrix}$$

Большая группа команд позволяет выделять из матрицы подструктуры: столбцы, строки, подматрицы и миноры. Для выделения строки с заданным номером следует применять команду `row(A,i)`, а чтобы получить нужный столбец можно использовать команду `col(A,j)`. Здесь `A` — матрица, а пара-

метры i и j являются номерами соответственно выделяемой строки или столбца. Формирование подматрицы, состоящей из элементов столбцов с номерами от i1 до i2 и строк с номерами от j1 до j2, осуществляется командой submatrix(A,i1..i2,j1..j2). Существует аналогичная команда для выделения вектора, состоящего из элементов с номерами от i1 до i2 исходного вектора, — subvector(vec,i1..i2). Матрица минора элемента с индексами (i,j) получается командой minor(A,i,j). Напомним, что минор получается из исходной матрицы вычеркиванием i-й строки и j-го столбца. Отметим, что все перечисленные команды не изменяют структуру исходной матрицы, а только выделяют из нее соответствующие подструктуры. Примеры использования этих команд приведены ниже.

```
> A:=matrix(4,3,[1, 1,-1,
                 1, 1,-1,
                 1, 1,-1,
                -1,-1,-1]);
```

$$A := \begin{bmatrix} 1 & 1 & -1 \\ 1 & 1 & -1 \\ 1 & 1 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

```
> row(A,4);
```

$$[-1, -1, -1]$$

```
> A1:=submatrix(A,2..3,2..3);
```

$$A1 := \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}$$

```
> A2:=minor(A,3,2);
```

$$A2 := \begin{bmatrix} 1 & -1 \\ 1 & -1 \\ -1 & -1 \end{bmatrix}$$

Пакет linalg содержит ряд команд, предназначенных для выполнения линейных преобразований над строками и столбцами исходной матрицы. Команда addcol(A,j1,j2,expr) создает новую матрицу из матрицы A путем прибавления к столбцу номер j1 столбца с номером j2, умноженного на значение параметра expr. Команда для выполнения аналогичных действий над строками матрицы имеет следующий синтаксис: addrow(A,i1,i2,expr). Умножить столбец с номером j на значение выражения expr можно командой mulcol(A,j,expr), а то же самое действие для строки номер i выполняется командой mulrow(A,i,expr). Для перестановки местами двух строк или двух столбцов матрицы A следует обратиться соответственно к командам swaprow(A,i1,i2) или swapcol(A,j1,j2). Эти команды полезны при приведении матрицы к треугольному виду или вычислении ее определителя разложением по строкам или столбцам.

Команды пакета linalg предоставляют возможность выполнения основных матричных и векторных операций: перемножение матриц, умножение матрицы на вектор, вычисление транспонированной и обратной матриц, а также вычисление определителя квадратной матрицы.

Сложение двух матриц осуществляется двумя способами: командой add() и командой evalm(). Отметим, что команда evalm() служит для вычисления матрицы или вектора на уровне их элементов и используется для вычисления любых возможных действий над матрицами, заданными операторами сложения (+), вычитания (-), умножения (&*), деления (/) и возведения в степень (^). Обратим внимание, что для выполнения некоммутативного умножения матриц используется операция умножения со знаком &*, а не знаком коммутативного умножения * чисел. Итак, сложить две матрицы одинаковой размерности можно либо командой evalm(A+B), либо командой add(A,B). Последнюю команду можно использовать для вычисления линейной комбинации двух матриц со скалярными множителями — add(A,B,scalar1,scalar2). В этом случае результатом будет матрица, вычисляемая по следующей формуле: scalar1*A+scalar2*B.

Умножение матрицы на матрицу (или вектор) выполняется командой multiply(A,B) или evalm(A&*B). При этом, естественно, размерности матриц-сомножителей должны быть таковыми, чтобы соответствующие операции существовали.

Возвести квадратную матрицу в степень n можно командой evalm(A^n). Для вычисления обратной матрицы к заданной можно воспользоваться либо командой inverse(A), либо командой evalm(1/M). Транспонированная матрица вычисляется с помощью обращения к команде transpose(A). Определитель и ранг матрицы можно получить, обратившись к командам det(A) и rank(A), соответственно.

Одной из важных характеристик квадратных матриц являются собственные числа и соответствующие им собственные векторы. Для вычисления собственных чисел и векторов числовой квадратной матрицы можно воспользоваться отложенной командой Eigenvals(A,V) основной библиотеки Maple. В ней первый параметр представляет собой матрицу, для которой вычисляются собственные числа, а второй необязательный параметр является матрицей, столбцы которой содержат собственные векторы, соответствующие собственным числам. Для получения результата применения этой отложенной команды следует использовать команду evalf(). Пример вычисления с ее помощью собственных чисел и собственных векторов представлен ниже.

```
> A:=matrix(3,3,[ 1, 2, 3,
                 -1,-2, 3,
                  0, 2, 9]);
```

$$A := \begin{bmatrix} 1 & 2 & 3 \\ -1 & -2 & 3 \\ 0 & 2 & 9 \end{bmatrix}$$

```
> evalf(Eigenvals(A,V));
```

$$[-1.905615188, .6683267973, 9.422305214]$$

```
> evalm(V);
```

$$\begin{bmatrix} .955495034 & -.5719289135 & .4022344932 \\ -.2869586740 & 1.203146218 & .2228332916 \\ .06843945323 & -.2174334239 & .9847493465 \end{bmatrix}$$

Для получения собственных чисел и векторов символьных матриц следует соответственно использовать команды `eigenvals(A)` и `eigenvects(A)`. Результатом выполнения первой команды будет массив, содержащий собственные числа матрицы, а результатом второй команды будет массив строк, элементами которых являются собственное число, его кратность и соответствующий собственному числу собственный вектор.

```
> A:=matrix(2,2,[ 0, 1,
                  a, b]);
```

$$A := \begin{bmatrix} 0 & 1 \\ a & b \end{bmatrix}$$

```
> eigenvals(A);
```

$$\frac{1}{2}b + \frac{1}{2}\sqrt{b^2+4a}, \frac{1}{2}b - \frac{1}{2}\sqrt{b^2+4a}$$

```
> eigenvects(A);
```

$$\left[\frac{1}{2}b + \frac{1}{2}\sqrt{b^2+4a}, 1, \left\{\left[-\frac{\frac{1}{2}b - \frac{1}{2}\sqrt{b^2+4a}}{a}, 1\right]\right\}\right],$$

$$\left[\frac{1}{2}b - \frac{1}{2}\sqrt{b^2+4a}, 1, \left\{\left[-\frac{\frac{1}{2}b + \frac{1}{2}\sqrt{b^2+4a}}{a}, 1\right]\right\}\right]$$

В пакете есть специальная команда `linsolve()` решения системы линейных алгебраических уравнений. Эта команда решает матричное уравнение `Ax=b`. Матрица `A` и вектор `b` передаются ей в качестве параметров:

```
linalg(A, b, 'r',v);
```

Размерность вектора `b` должна равняться числу строк матрицы `A`. Неизвестная переменная `r`, если задана, содержит после решения ранг матрицы `A`. Если ранг матрицы `A` меньше количества неизвестных, определяемых коли-

чеством столбцов матрицы A, то решение представляется в параметрической форме. Если задан необязательный четвертый параметр v, представляющий имя, то параметры задаются в виде _v[1], _v[2], ..., иначе используется имя по умолчанию, равное символу (t). Результатом выполнения команды будет вектор, в случае существования решения, и пустая последовательность — в противном случае.

Эта команда может решать и матричное уравнение вида AX=B, в котором количество строк матрицы B равняется количеству строк матрицы A, количество столбцов матрицы B равняется количеству столбцов матрицы неизвестных X, а число строк матрицы X равняется числу столбцов матрицы A. В этом случае, по существу, решается последовательность уравнений A col(X,i)=col(B,i), где i пробегает значения от 1 до количества столбцов матрицы B. Результатом будет матрица, столбцы которой являются решением соответствующей линейной системы из указанной последовательности систем уравнений. Если решения не существует, то результатом будет пустая последовательность.

Пример 3.2. Решение систем линейных алгебраических уравнений

```
> A := matrix( [[1,2],[1,3]]):
> b := vector( [1,-2]):
> linsolve(A, b);
```

$$[7, -3]$$

```
> B := matrix( [[1,1],[-2,1]]):
> linsolve(A, B);
```

$$\begin{bmatrix} 7 & 1 \\ -3 & 0 \end{bmatrix}$$

```
> A := matrix( [[5,7],[0,0]]):
> b := vector( [3,0]):
> linsolve(A, b, 'r');
```

$$\left[\frac{3}{5} - \frac{7}{5}_t_1, _t_1\right]$$

```
> r; # Ранг матрицы A
```

$$1$$

```
> linsolve(A, b, 'r', par);
```

$$\left[\frac{3}{5} - \frac{7}{5}par_1, par_1\right]$$

Совет

Рекомендуется для ускорения расчетов при решении систем линейных алгебраических уравнений использовать команду `linsolve()` из пакета `linalg`, так как для систем линейных уравнений больших размеров эта команда выполняется быстрее, чем универсальная команда `solve()`.

Пакет `linalg` содержит более ста полезных команд выполнения матричных операций, а также структурных преобразований матриц. Перечень всех доступных команд можно найти на странице справки, отображаемой командой `?linalg`.

3.2.2. Пакет *LinearAlgebra*

Все команды пакета `LinearAlgebra` можно вызывать непосредственно по имени, предварительно подключив все его команды функцией

```
> with(LinearAlgebra);
```

или отдельную команду с использованием синтаксиса

```
> with(LinearAlgebra, имя_команды);
```

Можно вызывать команду, предварительно не подключая ее, а используя длинное имя

```
> LinearAlgebra[имя_команды](параметры);
> LinearAlgebra['имя_команды'](параметры);
```

Последняя форма (имя команды, заключенное в одинарные кавычки), вызывает соответствующую команду пакета, даже если в текущем сеансе используется какой-либо объект с таким же именем.

Пакет `LinearAlgebra` реализован в виде модуля, новой языковой конструкции Maple, реализующей элементы объектно-ориентированного программирования. Каждая команда является методом объекта `LinearAlgebra`, а поэтому ее можно вызывать, используя специальную операцию `:-` обращения к методу объекта:

```
> LinearAlgebra:-имя_команды(параметры);
```

В этом случае вызываемая команда также будет загружена, не конфликтуя с объектом другого типа, созданным в текущем сеансе.

Замечание

Создание и работа с объектами, реализуемыми в Maple в виде модулей, рассматривается в гл. 5.

Совет

Ограниченный объем книги не позволяют нам в полной мере описать возможности пакета `LinearAlgebra`, поэтому настоятельно рекомендуем обратиться к странице Справки по этому пакету, которую можно отобразить командой `?LAOverview`. На ней расположены ссылки на другие страницы, подробно описывающие работу и программирование пакета `LinearAlgebra`, включая рабочие листы с примерами использования подпрограмм пакета NAG.

3.2.2.1. Основные типы данных

Основные типы данных, с которыми работают команды пакета `LinearAlgebra`, являются скаляры, представляющие как числа, так и алгебраические выражения, а также матрицы и векторы, определяемые на базе нового типа данных r-таблицы. (Мы не будем рассматривать здесь этот структурный тип данных, а отошлем интересующегося читателя к Справке Maple.) Матрицы и векторы создаются с помощью соответствующих конструкторов.

Конструктором матриц является команда `Matrix()` (обязательно с заглавной буквы), синтаксис которой имеет следующий вид:

```
Matrix(r, c, init, ro, sc, sh, st, o, dt, f, a);
```

Семантика параметров и их допустимые значения представлены в табл. 3.2.

Таблица 3.2. Параметры конструктора матриц

Параметр	Описание
`r`	Неотрицательное целое или диапазон целых чисел, начинающийся с 1. Представляет количество строк в матрице
`c`	Неотрицательное целое или диапазон целых чисел, начинающийся с 1. Представляет количество столбцов в матрице
`init`	Задает значения элементов матрицы. Может быть одним из следующих объектов Maple: *процедурой*, входными параметрами которой является пара целых положительных чисел, определяющих индексы элемента, а возвращаемым значением величина этого элемента, например, `(i,j)->i*j`; *алгебраическим выражением*, которое вычисляется как процедура с двумя параметрами, возвращающая значение элемента; *таблицей*, элементы которой с неотрицательными индексами представляют значения соответствующих элементов матрицы; *множеством уравнений* вида `(i,j)=значение`, в которых неотрицательные индексы представляют индексы соответствующего элемента матрицы; *массивом* на основе таблицы или r-таблицы, созданным, соответственно, либо командой `array()`, либо командой `Array()`, у которого индексы начинаются с 1; *матрицей* на основе r-таблицы, т. е. матрицей, созданной конструктором `Matrix()`; *списком*, элементы которого интерпретируются как значения первой строки матрицы, или списком, элементами которого являются списки, интерпретируемые как последовательные строки матрицы
`ro`	Задается в виде `readonly` или `readonly=true` и определяет, что значения элементов матрицы, определенные при ее создании, не могут быть изменены в дальнейшем

Таблица 3.2 (окончание)

Параметр	Описание
sc	Уравнение вида scan=имя или scan=список, определяющее структуру и/или порядок данных при интерпретации начальных значений, задаваемых параметром init
sh	Уравнение вида shape=имя или shape=список, определяющее одну или более встроенных или пользовательских индексных функций, задающих расположение в памяти элементов матрицы
st	Уравнение вида storage=имя, где имя является одним из допустимых режимов памяти, определяя тем самым требования памяти для размещения элементов матрицы
o	Уравнение вида order=имя, где имя может быть либо C_order, либо Fortran_order, задавая хранение матрицы в памяти, соответственно, по строкам или столбцам
dt	Уравнение вида datatype=имя, где имя может быть любым типом Maple, определяющим тип данных, хранимых в матрице
f	Уравнение вида fill=значение, определяющее значение, присваиваемое неопределенным элементам матрицы. По умолчанию оно равно 0
a	Уравнение вида attributes=список, определяющее атрибуты (положительно-определенная, эрмитова и т. д.), с которыми матрица была создана

Все параметры являются необязательными, и в случае их отсутствия создается матрица размерности 0×0. Собственно говоря, для создания матрицы важны первые три параметра. Остальные используются различными командами для ускорения ее обработки.

Пример 3.3. Создание матриц

```
> Matrix(2);
```

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

```
> Matrix(2,3);
```

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

```
> Matrix(1..1,1..4,6);
```

$$[6 \quad 6 \quad 6 \quad 6]$$

```
> Matrix([[1,2,3],[4,5,6]]);
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

```
> Matrix(2,(i,j)->x^(i+j));
```

$$\begin{bmatrix} x^2 & x^3 \\ x^3 & x^4 \end{bmatrix}$$

Создать вектор можно конструктором `Vector()` со следующим синтаксисом:

```
Vector(d, init, ro, sh, st, dt, f, a, o);
Vector[column](d, init, ro, sh, st, dt, f, a, o);
Vector[row](d, init, ro, sh, st, dt, f, a, o);
```

В пакете `LinearAlgebra` различаются векторы-столбцы, задаваемые с помощью первых двух форм конструктора, и векторы-строки, для задания которых служит третья форма конструктора. Их можно определять только с помощью первой формы, задавая соответствующее значение последнего параметра `o`: `column` или `row`. Первый параметр `d` задает размерность вектора и может принимать только целые положительные значения, большие или равные 1. Остальные параметры соответствуют аналогичным в конструкторе матриц.

Пример 3.4. Создание векторов

```
> Vector(2);
```

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

```
> Vector(1..3,5,orientation=row);
```

$$[5, 5, 5]$$

```
> Vector[row]([1,2,3]);
```

$$[1, 2, 3]$$

```
> Vector(2,(i)->x^i);
```

$$\begin{bmatrix} x \\ x^2 \end{bmatrix}$$

При интерактивной работе в Maple иногда не совсем удобно создавать матрицы и векторы, обращаясь к их конструктору. Разработчики пакета `LinearAlgebra` предоставили пользователю возможность использования краткой формы задания векторов и матриц:

- `<a,b,c>` создает матрицу или вектор по строкам;
- `<a|b|c>` создает матрицу или вектор по столбцам.

Если величины, задаваемые в угловых скобках, не являются скалярами, то создается матрица, в противном случае вектор.

Пример 3.5. Краткая форма задания векторов и матриц

```
> V1:=<1,2,3>; # Создание вектора-столбца
```

$$V1 := \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
> V2:=<1|2|3>; # Создание вектора-строки
```

$$V2 := [1, 2, 3]$$

```
> M1:=<<1|2>,<3|4>>; # Создание матрицы по строкам
```

$$M1 := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
> M2:=<<1,3>|<2,4>>; # Создание матрицы по столбцам
```

$$M2 := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
> <M2|M1>; # Создание матрицы из двух других
```

$$\begin{bmatrix} 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 \end{bmatrix}$$

В завершение разговора о задании матриц и векторов следует сказать о специальных типах матриц, поддерживаемых пакетом `LinearAlgebra`.

Для создания специальных типов матриц и векторов — единичных, нулевых, матриц и векторов констант и скалярных — можно использовать специальные конструкторы, хотя объекты указанных типов можно создать и при помощи общих конструкторов. Пример 3.6 демонстрирует все типы специальных конструкторов.

Пример 3.6. Специальные типы векторов и матриц

```
> IdentityMatrix(2,2); # Единичная матрица
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

```
> ZeroMatrix(2,3); # Нулевая матрица
```

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

```
> ConstantMatrix(6,2); # Матрица-константа
```

$$\begin{bmatrix} 6 & 6 \\ 6 & 6 \end{bmatrix}$$

```
> ScalarMatrix(a^2,2); # Скалярная матрица
```

$$\begin{bmatrix} a^2 & 0 \\ 0 & a^2 \end{bmatrix}$$

```
> UnitVector[row](2,3); # Единичный вектор
```

$$[0,1,0]$$

```
> ZeroVector[row](3); # Нулевой вектор
```

$$[0,0,0]$$

```
> ConstantVector[row](5,3); # Вектор-константа
```

$$[5,5,5]$$

```
> ScalarVector[row](x^2+y^2,3,4); # Скалярный вектор
```

$$[0,0,x^2+y^2,0]$$

При задании матриц и векторов больших размеров они не отображаются на рабочем листе. Вместо их содержимого отображается подсказка, что здесь расположен соответствующий объект и указывается его структура и размерность:

```
> Matrix(15,15,(i,j)->i*j);
```

$$\begin{bmatrix} \text{15 x 15 Matrix} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran_order} \end{bmatrix}$$

Для просмотра подобных векторов и матриц в Maple 6 включена специальная программа просмотра структурированных данных (Structured Data Browser), которую можно вызвать из контекстного меню командой **Browser**. Окно этой программы представлено на рис. 3.1.

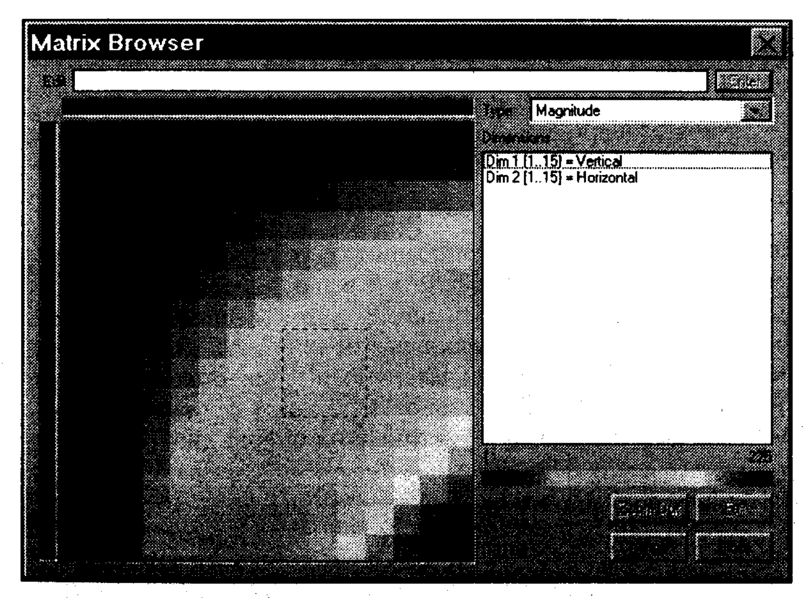


Рис. 3.1. Окно программы просмотра матриц и векторов больших размерностей

В этом окне отображается вся матрица, элементы которой представлены квадратиками разных цветов, в зависимости от величины значения. Можно установить режим отображения нулевых элементов белым цветом, а ненулевых черным (значение **Structure** в списке **Type**) или вместо умалчиваемой цветной легенды (значение **Magnitude** в списке **Type**) установить градации серого (значение **Density** в списке **Type**). Выделив мышью необходимые элементы матрицы в окне отображения, как показано на рис. 3.1, в том же окне отобразятся их значения, которые можно корректировать (рис. 3.2).

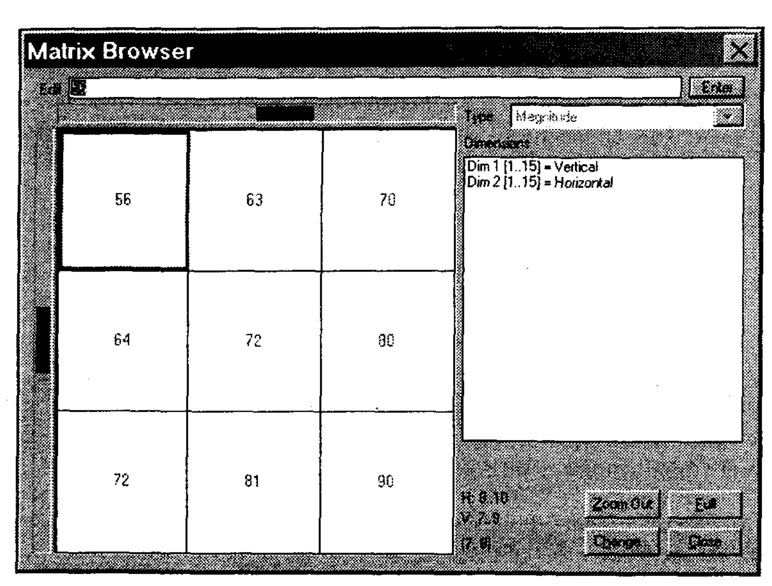


Рис. 3.2. Редактирование данных в программе просмотра матриц и векторов больших размерностей

3.2.2.2. Элементарные операции с матрицами и векторами

Как уже отмечалось ранее, основные операции с матрицами в пакете `LinearAlgebra` выполняются проще, чем такие же в пакете `linalg`. Это связано с тем, что идентификаторы векторов и матриц здесь вычисляются не до уровня имени, а непосредственно до уровня вычисления их компонентов. В связи с этим возможно выполнение поэлементного сложения, вычитания и составления линейных комбинаций векторов и матриц одинаковой размерности с использованием обычных арифметических операций.

Пример 3.7. Элементарные операции с векторами и матрицами

```
> M1:=<<1|2>,<3|4>>;
```

$$M1 := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
> M2:=<<10|7>,<8|15>>;
```

$$M2 := \begin{bmatrix} 10 & 7 \\ 8 & 15 \end{bmatrix}$$

```
> M1+M2;
```

$$\begin{bmatrix} 11 & 9 \\ 11 & 19 \end{bmatrix}$$

```
> M1-M2;
```

$$\begin{bmatrix} -9 & -5 \\ -5 & -11 \end{bmatrix}$$

```
> 3.1*M1+5*M2;
```

$$\begin{bmatrix} 53.1000000000000014 & 41.2000000000000029 \\ 49.2999999999999972 & 87.4000000000000057 \end{bmatrix}$$

```
> V1:=<1|4>;
```

$$V1 := [1, 4]$$

```
> V2:=<3|8>;
```

$$V2 := [3, 8]$$

```
> 3*V1-6*V2;
```

$$[-15, -36]$$

```
> V3:=<3,8>;
```

$$V3 := \begin{bmatrix} 3 \\ 8 \end{bmatrix}$$

```
> V1+V3; # Нельзя получить линейную комбинацию
         # вектора-строки и вектора-столбца
Error, (in rtable/Sum) invalid arguments
```

Если складывается скаляр с матрицей, то это равносильно сложению матрицы с единичной матрицей, элементы которой умножены на заданный скаляр, вектор нельзя складывать со скаляром:

```
> 10 + <<2|5|11>,<4|6|7>>;
  10*IdentityMatrix(2,3) + <<2|5|11>,<4|6|7>>;
```

$$\begin{bmatrix} 12 & 5 & 11 \\ 4 & 16 & 7 \end{bmatrix}$$

$$\begin{bmatrix} 12 & 5 & 11 \\ 4 & 16 & 7 \end{bmatrix}$$

```
> 2+<1,2>;
Error, (in rtable/Sum) invalid arguments
```

Замечание

Построение линейной комбинации матриц и векторов можно также выполнить, используя, соответственно, команды `MatrixAdd()` **и** `VectorAdd()`.

Так как произведение матриц (имеется в виду операция скалярного умножения) не является коммутативной, то использование операции коммутативного умножения (*) для векторов и матриц приводит к ошибке. (Исключение допускается только для умножения матрицы саму на себя, причем в этом случае выполняется операция скалярного умножения.) Коммутативное умножение можно использовать для перемножения скаляра и матрицы или вектора. В этом случае все элементы этих объектов умножаются на соответствующий скаляр:

```
> -3*<1|2|3>;
```

$$[-3, -6, -9]$$

```
> 4*<<7,8>|<1,6*t>>;
```

$$\begin{bmatrix} 28 & 4 \\ 32 & 24\,t \end{bmatrix}$$

Однако если скаляр содержит неопределенную переменную, то перемножения не происходит, так как Maple не знает, какой объект в дальнейшем эта переменная может содержать. Для выполнения такого умножения следует использовать команду `simplify()` с параметром `symbolic` или опцией `assume=scalar`:

```
> mull:=x^2*<1|2|3>;
```

$$mull := x^2\,[1, 2, 3]$$

```
> simplify(mull,symbolic);
```

$$[x^2, 2\,x^2, 3\,x^2]$$

```
> simplify(mull,assume=scalar);
```

$$[x^2, 2\,x^2, 3\,x^2]$$

Выполнить некоммутативное умножение в Maple 6 можно операцией, символом которой является точка (.). Она никогда не меняет сомножители местами, поэтому произведения `x.y.z` и `x.z.y` не являются тождественными:

```
> m1:=x.y.z;
```

$$m1 := x\,.\,y\,.\,z$$

```
> m2:=x.z.y;
```

$$m2 := x\,.\,z\,.\,y$$

```
> if(m1=m2) then
      print("true")
  else
      print("false")
  end if;
```

"false"

В этом примере нами использован оператор условия языка программирования Maple, чтобы показать не тождественность результата операции неком-

мутативного умножения трех величин, взятых в разных порядках. Надеемся, что читателю понятен смысл этого оператора, хотя при желании он может обратиться к главе 6, посвященной именно программированию в Maple.

Внимание!

В Maple 6 операция точка (.) больше не является операцией конкатенации, как это было в предыдущих версиях Maple. Для конкатенации используется операция (||).

Эта же операция, примененная к матрицам и векторам выполняет их скалярное произведение.

Пример 3.8. Скалярные произведения векторов и матриц

```
> <1,3>.<4|6>; # Вектор-столбец умножается на вектор-строку
```

$$\begin{bmatrix} 4 & 6 \\ 12 & 18 \end{bmatrix}$$

```
> <4|6>.<1,3>; # Вектор-строка умножается на вектор-столбец
```

$$22$$

```
> <<3,-1>|<-8,15>|<9,10>>.<<1,x,y>|<4,-7,2>>; # Матрица 2x3 умножается
                                              # на матрицу 3x2
```

$$\begin{bmatrix} 3-8x+9y & 86 \\ -1+15x+10y & -89 \end{bmatrix}$$

Для получения степени квадратной матрицы можно последовательно применить операцию скалярного умножения необходимое число раз или операцию возведения в степень (^):

```
> M:=<<0.2|0.8>,<0.7|0.5>>;
```

$$M := \begin{bmatrix} .2 & .8 \\ .7 & .5 \end{bmatrix}$$

```
> M.M.M.M.M;
```

$$\begin{bmatrix} .679599999999999982 & .902320000000000011 \\ .789529999999999842 & 1.01797000000000004 \end{bmatrix}$$

```
> M^5;
```

$$\begin{bmatrix} .679599999999999982 & .902320000000000122 \\ .789529999999999842 & 1.01797000000000004 \end{bmatrix}$$

Показатель степени может быть и отрицательным целым числом, что позволяет вычислять обратную матрицу и ее степени:

```
> M^(-1);
```

$$\begin{bmatrix} -1.08695652173913060 & 1.73913043478260887 \\ 1.52173913043478270 & -.434782608695652272 \end{bmatrix}$$

```
> %.M;
```

$$\begin{bmatrix} 1. & -.111022302462515654\ 10^{-15} \\ -.555111512312578272\ 10^{-16} & 1. \end{bmatrix}$$

Для выделения элементов матрицы и ее подматриц используется индексная запись, причем в качестве индекса можно применять диапазон, что позволяет выделять целые блоки исходной матрицы:

```
> M:=Matrix(5,(i,j)->3*i-j);
```

$$M := \begin{bmatrix} 2 & 1 & 0 & -1 & -2 \\ 5 & 4 & 3 & 2 & 1 \\ 8 & 7 & 6 & 5 & 4 \\ 11 & 10 & 9 & 8 & 7 \\ 14 & 13 & 12 & 11 & 10 \end{bmatrix}$$

```
> M[3,2];                      # Выбор элемента матрицы
```

$$7$$

```
> M[3,1..-1];                  # Выбор третьей строки
```

$$[8, 7, 6, 5, 4]$$

```
> M[1..-1,2];                  # Выбор второго столбца
```

$$\begin{bmatrix} 1 \\ 4 \\ 7 \\ 10 \\ 13 \end{bmatrix}$$

```
> M[2..3,2..4];                # Выбор блока размерности 2x3
```

$$\begin{bmatrix} 4 & 3 & 2 \\ 7 & 6 & 5 \end{bmatrix}$$

Присваивание новых значений элементам матрицы также осуществляется с помощью индексов, причем и здесь можно использовать диапазон. Главное, чтобы размерность матрицы в левой части оператора присваивания соответствовала размерности матрицы в правой части.

Возможности выбора и присваивания значений целым строкам или столбцам используется для осуществления операций со строками или столбцами матрицы.

Пример 3.9. Операции со строками матриц

```
> M:=Matrix(3,(i,j)->i*5+j-5);
```

$$M := \begin{bmatrix} 1 & 2 & 3 \\ 6 & 7 & 8 \\ 11 & 12 & 13 \end{bmatrix}$$

```
> M[[1,2],1..-1] := M[[2,1],1..-1]; # Перестановка местами строк 1 и 2
```

$$M_{[1,2],1..-1} := \begin{bmatrix} 6 & 7 & 8 \\ 1 & 2 & 3 \end{bmatrix}$$

```
> M;
```

$$\begin{bmatrix} 6 & 7 & 8 \\ 1 & 2 & 3 \\ 11 & 12 & 13 \end{bmatrix}$$

```
> M[3,1..-1] := 3*M[3,1..-1]; # Умножение строки 3 на число 3
```

$$M_{3,1..-1} := [33, 36, 39]$$

```
> M;
```

$$\begin{bmatrix} 6 & 7 & 8 \\ 1 & 2 & 3 \\ 33 & 36 & 39 \end{bmatrix}$$

```
> M[3,1..-1] := M[3,1..-1] -16*M[2,1..-1]; # Вычитание из строки 3
                                           # строки 2, умноженной на 16
```

$$M_{3,1..-1} := [17, 4, -9]$$

```
> M;
```

$$\begin{bmatrix} 6 & 7 & 8 \\ 1 & 2 & 3 \\ 17 & 4 & -9 \end{bmatrix}$$

Все перечисленные операции можно выполнять с помощью команд пакета `LinearAlgebra`, которые рекомендуется использовать при программировании в Maple, хотя и допустимо их использование при интерактивной работе. Неполный список команд допустимых операций над матрицами и векторами приведен в табл. 3.3.

Таблица 3.3. *Команды выполнения операций над матрицами и векторами*

Название команды	Описание
`DeleteRow`	Удаление строки матрицы
`DeleteColumn`	Удаление столбца матрицы
`Row`	Выделение строки матрицы
`Column`	Выделение строки матрицы
`SubMatrix`	Выделение подматрицы из заданной матрицы
`SubVector`	Выделение подвектора из заданного вектора
`ScalarMultiply`	Умножение матрицы/вектора на скаляр
`MatrixVectorMultiply`	Скалярное произведение матрицы на вектор-столбец

Таблица 3.3 (окончание)

Название команды	Описание
VectorMatrixMultiply	Скалярное произведение вектора-строки на матрицу
MatrixMatrixrMultiply	Скалярное произведение матрицы на матрицу
MatrixInverse	Вычисление обратной матрицы
Determinant	Вычисление определителя матрицы
Minor	Выделение миноров матрицы
ConditionNumber	Вычисление числа обусловленности матрицы
Eigenvalues	Вычисление собственных значений матрицы
Eigenvectors	Вычисление собственных векторов

Замечание

Получить полную информацию о всех перечисленных в табл. 3.3 командах можно на страницах справки по каждой из команд, которые отображаются после выполнения команды `?имя_команды`.

3.2.2.3. Решение систем линейных уравнений

В пакет `LinearAlgebra`, как и в пакет `linalg`, входит специальная команда `LinearSolve()` решения систем линейных алгебраических уравнений. В отличие от своего двойника `linsolve()` из пакета `linalg` в этой команде можно указать способ, которым следует решать систему, но аналогично своему двойнику она требует задания системы уравнений в матричной форме, т. е. в качестве параметров ей передаются матрица `A` системы и вектор правых частей `b`. Сама система в матричной форме записывается в виде `A.x=b`.

Общий синтаксис команды `LinearSolve()` следующий:

```
LinearSolve(A, B, m, t, c, ip, outopts);
```

Здесь параметр `A` представляет матрицу системы, а параметр `B` — ее левую часть, причем `B` может задаваться как в виде вектора, так и в виде матрицы. В последнем случае за одно обращение к команде решения системы линейных уравнений будет решаться множество систем с правыми частями, представленными векторами — столбцами матрицы `B`. Параметр `B` можно и не задавать, передавая в качестве первого параметра расширенную матрицу системы `<A|B>`. Размерности матрицы решения согласовываются с размерностями матриц правой и левой частей уравнения. Если матрица системы `A` имеет размерность `mxn`, и правая часть представлена матрицей `mxp`, то результатом будет матрица размерности `nxp`, столбцы которой будут являться решениями соответствующих систем.

Необязательный параметр m задается в форме уравнения method=имя, где имя выбирается из следующего списка: 'none', 'solve', 'subs', 'Cholesky', 'LU', 'QR', 'SparseLU' и определяет метод решения системы уравнений.

Параметр t, который также необязателен, определяет базовое имя переменной в форме free=имя, которое используется для конструирования имен параметров в случае, если исходная система линейных уравнений имеет множество решений.

Необязательным параметром c в форме уравнения conjugate=true/false определяется, следует ли строить эрмитову сопряженную матрицу при использовании метода Холецкого или QR-декомпозиции.

Параметр in задается в виде inplace=true/false и определяет, помещать ли решение в вектор или матрицу B, или формировать новый объект для решения. Значение по умолчанию false.

Последним параметром outopts задаются опции outputoptions, предоставляющие дополнительную информацию конструктору решения (неизменяемая матрица, тип, математические атрибуты и т. д.).

Несколько примеров решения систем линейных уравнений с помощью команд пакета LinearAlgebra помогут освоиться в его использовании. Следует сказать, что пакет позволяет использовать для числовых матриц с вещественными числами в форме с плавающей точкой либо программную эмуляцию вычислений с вещественными числами с произвольным числом значащих цифр в мантиссе, либо использовать вещественную арифметику процессора, что, естественно, приводит к увеличению скорости вычислений и особенно эффективно при работе с большими числовыми системами линейных уравнений и числовыми матрицами больших размерностей. При использовании любой из двух реализаций вычислений с вещественными числами для выполнения необходимых действий подключаются откомпилированные программы из пакета численных вычислений NAG. Выбор соответствующей арифметики Maple осуществляет автоматически, но можно и явно указать, какую из них использовать, задав значение true системной переменной UseHardwareFloats для использования арифметики процессора и false — в противном случае. В примерах этого раздела мы везде подразумеваем, что используется программная эмуляция арифметики чисел с плавающей точкой, т. е. UseHardwareFloats=false.

Решим следующую систему уравнений

$$\begin{bmatrix} 0 & 3 & 1 \\ 7 & -13 & -2 \\ 1 & 2 & 4 \end{bmatrix} . X = \begin{bmatrix} 5.84 \\ -8.46 \\ 13.11 \end{bmatrix}$$

методом LU-декомпозиции. Это можно сделать за одно обращение к команде LinearSolve() или по шагам: сначала осуществить LU-разложение матрицы системы, а потом построить решение.

Решение за одно обращение представлено в примере 3.10.

Пример 3.10. Решение системы методом LU-декомпозиции

```
> with(LinearAlgebra):
  UseHardwareFloats:=false;
  Digits:=30:
```

$$UseHardwareFloats := false$$

```
> M:=<<0|3|1>,<7|-13|-2>,<1|2|4>>;
```

$$M := \begin{bmatrix} 0 & 3 & 1 \\ 7 & -13 & -2 \\ 1 & 2 & 4 \end{bmatrix}$$

```
> V:=<5.84,-8.46,13.11>;
```

$$V := \begin{bmatrix} 5.84 \\ -8.46 \\ 13.11 \end{bmatrix}$$

```
> x := LinearSolve(M, V, method='LU');
  x := LinearSolve(<M|V>, method='LU');
```

$$x := \begin{bmatrix} 1.64999999999999999999999999999 \\ 1.18999999999999999999999999999 \\ 2.27000000000000000000000000002 \end{bmatrix}$$

$$x := \begin{bmatrix} 1.64999999999999999999999999999 \\ 1.18999999999999999999999999999 \\ 2.27000000000000000000000000002 \end{bmatrix}$$

```
> M.x-V;
```

$$\begin{bmatrix} -.1\ 10^{-28} \\ -.4\ 10^{-28} \\ 0. \end{bmatrix}$$

Пример 3.10 демонстрирует, что можно увеличить при использовании программной эмуляции арифметики действительных чисел с плавающей точкой точность вычислений, установив необходимое количество цифр в мантиссе в системной переменной `Digits`. В нем же показано два способа задания уравнения в команде `LinearSolve()`: через передачу в нее матрицы системы и вектора правой части посредством двух параметров и в виде одного параметра, представляющего расширенную матрицу системы. Последний оператор примера осуществляет проверку полученного решения. Как видим, результат вычитания левой и правой частей практически равен нулевому вектору, что говорит об истинности полученного решения.

Пошаговое решение той же системы методом декомпозиции представлено в примере 3.11.

Пример 3.11. Пошаговое решение системы методом LU-декомпозиции

```
> P,L,U:=LUDecomposition(M);
```

$$P, L, U := \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{1}{7} & \frac{9}{7} & 1 \end{bmatrix}, \begin{bmatrix} 7 & -13 & -2 \\ 0 & 3 & 1 \\ 0 & 0 & 3 \end{bmatrix}$$

```
> V2:=Transpose(P).V;
```

$$V2 := \begin{bmatrix} -8.46 \\ 5.84 \\ 13.11 \end{bmatrix}$$

```
> V3:=ForwardSubstitute(L,V2);
```

$$V3 := \begin{bmatrix} -8.460000000000000000000000000000 \\ 5.840000000000000000000000000000 \\ 6.810000000000000000000000000005 \end{bmatrix}$$

```
> x:=BackwardSubstitute(U,V3);
```

$$x := \begin{bmatrix} 1.649999999999999999999999999999 \\ 1.189999999999999999999999999999 \\ 2.270000000000000000000000000002 \end{bmatrix}$$

Сначала обращением к команде `LUDecomposition()`, единственным параметром которой является матрица системы `M`, получается ее разложение на три матрицы: `P`, `L` и `U`, причем матрица `P` представляет собой матрицу перестановок, а матрицы `L` и `U` являются, соответственно, нижней и верхней диагональной. Таким образом, нашу систему можно записать в виде `P.L.U.x=V`. Так как `P` — матрица перестановок, то ее обратная матрица равняется транспонированной. Умножив и левую, и правую части уравнения на матрицу, обратную `P` (вычисляется командой `Transpose(P)`), наша система преобразуется к виду `L.U=V2`, где вектор `V2` есть произведение транспонированной матрицы `P` на вектор правой части уравнения. Так как матрица `L` является нижней диагональной, то можно найти вектор `V3`, удовлетворяющий системе `L.V3=V2`, простой последовательной подстановкой вперед: первый элемент вектора выражается сразу же из первого уравнения указанной системы, затем его значение подставляется во второе уравнение и находится значение второго компонента вектора решения `V3` и т. д. В пакете `LinearAlgebra` этот метод реализуется командой `ForwardSubstitute()`. Теперь, так как `L.V3=V2` и `L.U.x=V2`, то для нахождения решения исходной системы уравнений достаточно найти решение системы `U.x=V3`, которое находится подстановкой назад в связи с тем, что матрица `U` является верхней диагональной: последний элемент вектора выражается сразу же из последнего уравнения указанной системы, затем его значение подставляется в пред-

последнее уравнение и находится значение предпоследнего компонента вектора решения x и т. д. Эта подстановка реализуется командой `BackwardSubstitute(U,V3)`. В результате без обращения к команде `LinearAlgebra` мы решили исходную систему уравнений методом LU-декомпозиции.

Замечание

При задании в качестве метода решения системы линейных уравнений в команде `LinearSolve()` метод LU-декомпозиции неявно выполняются все, выполненные нами в примере 3.11, действия.

Замечание

Можно выполнить подобные пошаговые вычисления при решении системы линейных уравнений и любым другим способом, реализованном в пакете `LinearAlgebra`.

3.2.2.4. Вычисления с использованием программ пакета NAG

В этом заключительном подразделе описания нового пакета `LinearAlgebra` мы сконцентрируем внимание читателя на проблеме численных расчетов с его помощью. В процессе решения задач линейной алгебры в среде пакета `LinearAlgebra` он автоматически выбирает, какую модель вычислений следует использовать: символьную, программную реализацию арифметики чисел с плавающей точкой с произвольным числом значащих цифр в мантиссе или арифметику чисел с плавающей точкой, поддерживаемую процессором компьютера, что, естественно, связано с ограничением числа значащих цифр в мантиссе.

Символьная модель самая медленная, тогда как процессорная модель самая быстрая. Более того, в зависимости от используемой модели для решения одной и той же задачи используются разные встроенные процедуры. Для символьных вычислений применяются интерпретируемые процедуры, написанные на языке Maple, и это приводит к снижению скорости вычислений, особенно ощутимой для матриц больших размеров. Для двух других моделей подключаются откомпилированные программы из пакета численных расчетов NAG, причем для программного моделирования вычислений с произвольным числом значащих цифр в мантиссе скорость вычислений меньше, чем при использовании арифметики процессора, хотя в последнем случае страдает точность вычислений. Решение задач с использованием программ NAG в любом случае быстрее, чем с интерпретируемыми процедурами Maple.

Как же Maple определяет, какую модель применять в конкретных случаях и, соответственно, какие процедуры следует использовать? Прежде всего опре-

деляется, содержат ли матрицы элементы, значениями которых являются числа с плавающей точкой и/или символы. Для этого он проверяет тип данных матрицы, определяемый опцией `datatype` конструктора при создании матрицы. Если она не была задана, то осуществляется поэлементная проверка типов данных содержимого матрицы, причем для больших матриц этот процесс может оказаться достаточно длительным. Поэтому рекомендуется задавать тип данных матрицы при ее создании.

После проверки происходит выбор подходящей модели в соответствии со следующим алгоритмом:

- если элементы матрицы содержат только числовые данные, причем хотя одно из них представлено числом с плавающей точкой, и значение переменной окружения `UseHardwareFloats` равно `true` (значение по умолчанию), используется арифметика процессора;
- если элементы матрицы содержат только числовые данные, причем хотя одно из них представлено числом с плавающей точкой, и значение переменной окружения `UseHardwareFloats` равно `false`, используется программная реализация арифметики вещественных чисел;
- если есть хоть одно не числовое значение, например, $\sqrt{2}$ или π, то используются процедуры символьных вычислений;
- если нет чисел с плавающей точкой, то используются процедуры символьных вычислений.

Для того чтобы увидеть, действительно ли будут вызываться процедуры пакета NAG и использоваться арифметика процессора, следует в специальной таблице `infolevel`, хранящей установки для уровня отображения информации пользователю при выполнении некоторых команд, установить для пакета `LinearAlgebra` уровень 1. Для этого следует использовать синтаксическую конструкцию

```
> infolevel[LinearAlgebra]:=1;
```

Вообще, можно устанавливать соответствующий уровень отображения информации для пользователя либо при выполнении определенной команды, например, `dsolve()`, либо всего пакета, либо для всех команд, задав в квадратных скобках вышеуказанной конструкции `all`:

```
> infolevel[dsolve]:=2;
```

$$infolevel_{dsolve} := 2$$

```
> infolevel[all]:=1;
```

$$infolevel_{all} := 1$$

Всего может быть пять уровней. Первый отображает сообщения для пользователя, второй и третий — общую информацию, включая информацию об

используемой методике или алгоритме, четвертый и пятый — более подробную информацию о процессе решения задачи.

Замечание

Не все команды отображают информацию, даже при установленном пятом уровне.

При использовании арифметики процессора отображаются имена программ пакета NAG с префиксом `hw_`, тогда как при программной реализации арифметики чисел с плавающей точкой эти программы имеют префикс `sw_`. Пример 3.12 демонстрирует использование вычислений с арифметикой процессора.

Пример 3.12. Вычисления с арифметикой процессора

```
> with(LinearAlgebra):
  infolevel[LinearAlgebra]:=1;
```

$$infolevel_{LinearAlgebra} := 1$$

```
> M1:=<<1|2>,<4|5>>;
```

$$M1 := \begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix}$$

```
> M1^(-1);
```

$$\begin{bmatrix} \frac{-5}{3} & \frac{2}{3} \\ \frac{4}{3} & \frac{-1}{3} \end{bmatrix}$$

```
> M2:=<<1.|2>,<4|5>>;
```

$$M2 := \begin{bmatrix} 1. & 2 \\ 4 & 5 \end{bmatrix}$$

```
> M2^(-1);
MatrixInverse:        "calling external function"
MatrixInverse:        "NAG"  hw_f07adf
MatrixInverse:        "NAG"  hw_f07ajf
```

$$\begin{bmatrix} -1.66666666666666666666666666666 & .666666666666666666666666666665 \\ 1.33333333333333333333333333333 & -.333333333333333333333333333332 \end{bmatrix}$$

При вычислении обратной матрицы `M1` использовались символьные процедуры, так как ни один ее элемент не является числом с плавающей точкой. Матрица `M2` содержит один элемент, представленный числом с плавающей точкой, а так как по умолчанию переменная окружения `UseHardwareFlots` имеет значение `true`, то при вычислении ее обратной использовались программы пакета NAG для вычислений с арифметикой процессора.

Замечание

Рекомендации по эффективному использованию памяти при работе с программами пакета NAG можно найти в примере, который вызывается командой `?examples,LA_NAG`.

3.3. Обыкновенные дифференциальные уравнения

Мы знакомы с универсальной командой решения обыкновенных дифференциальных уравнений `dsolve()`. С ее помощью можно получать общее решение дифференциального уравнения или системы дифференциальных уравнений, решать задачи Коши и краевые задачи. Эта команда всегда стремится найти общее решение в аналитическом виде и использовать его для построения решения краевой задачи или задачи Коши. Однако не всегда удается для обыкновенного дифференциального уравнения найти общее решение в замкнутой форме, и более того, существуют дифференциальные уравнения, для которых вообще невозможно построить общее решение в аналитическом виде. В таком случае прибегают к приближенным методам решения, которые реализуются той же командой `dsolve()` с соответствующими опциями (в форме рядов, с использованием численных методов типа Рунге-Кутта различной точности). После чего полученное в виде процедуры Maple решение можно использовать для построения таблицы его значений или отобразить в форме графика командой `odeplot()` из пакета графических команд `plots()`, о котором пойдет речь в гл. 4.

Пример 3.13. Численное решение системы дифференциальных уравнений

```
> dif1:=diff(x(t),t)=x(t)-y(t);
```

$$dif1 := \frac{\partial}{\partial t}\mathrm{x}(t) = \mathrm{x}(t) - \mathrm{y}(t)$$

```
> dif2:=diff(y(t),t)=x(t)+y(t);
```

$$dif2 := \frac{\partial}{\partial t}\mathrm{y}(t) = \mathrm{x}(t) + \mathrm{y}(t)$$

```
> inicond:=x(0)=1,y(0)=1;
```

$$inicond := \mathrm{x}(0) = 1, \mathrm{y}(0) = 1$$

```
> sol:=dsolve({dif1,dif2,inicond},{x(t),y(t)}, type=numeric);
```

$$sol := \mathbf{proc}(rkf45_x) \ \ldots \ \mathbf{end\ proc}$$

```
> sol(0);sol(1);sol(2);sol(3);
```

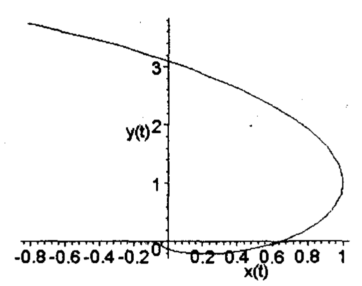
$$[t = 0, \mathrm{x}(t) = 1., \mathrm{y}(t) = 1.]$$

$$[t = 1, \mathrm{x}(t) = -.818661372311693780, \mathrm{y}(t) = 3.75604922293629252]$$

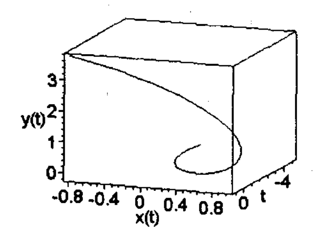
$$[t = 2, \mathrm{x}(t) = -9.79378206300327570, \mathrm{y}(t) = 3.64391727181316627]$$

$$[t = 3, \mathrm{x}(t) = -22.7190017594013122, \mathrm{y}(t) = -17.0500600314509967]$$

```
> with(plots):
> # Фазовый портрет системы
> odeplot(sol,[x(t),y(t)],-5..1, labels=["x(t),"y(t)],color=black);
```




```
> # Решение системы в пространстве
> odeplot(sol,[t,x(t),y(t)],-5..1,
        labels=["t","x(t),"y(t)],color=black);
```



Замечание

Система дифференциальных уравнений примера 3.13, являясь линейной системой дифференциальных уравнений, имеет общее аналитическое решение. Она взята нами исключительно в иллюстративных целях, чтобы показать возможности команды `pdeplot()`.

Оказывается, для некоторых дифференциальных уравнений реализованные в команде `dsolve()` приближенные методы не дают удовлетворительного результата в связи с накоплением погрешности при их использовании. (Мы не будем приводить примеры вычислительной неустойчивости численного решения дифференциального уравнения, но такое действительно случается и достаточно часто в практике моделирования реальных явлений.) В таких случаях приходится либо разрабатывать специальные методы приближенного интегрирования дифференциального уравнения, либо упрощать математическую модель явления, вводя некоторые ограничения, либо пытаться преобразовать дифференциальное уравнение, приведя его к виду, для которого можно построить удовлетворительное приближенное решение.

Пакет `DEtools` содержит набор команд для преобразования обыкновенных дифференциальных уравнений к специальному виду, для исследования гамильтоновых систем, для применения аппарата алгебр Ли при интегрирова-

нии дифференциальных уравнений, для работы с дифференциальными операторами и для построения различных видов графиков решения дифференциального уравнения и систем дифференциальных уравнений. Все перечисленные возможности пакета, кроме последней, с большой степенью вероятности представляют интерес для специалистов в области дифференциальных уравнений, тогда как возможность отображения решения дифференциального уравнения без явного построения численного решения в виде процедуры Maple представляет интерес для большого круга технических специалистов, работающих в разных областях. В этом разделе мы и остановимся только на графических возможностях пакета `DEtools`.

Замечание

Желающие изучить все возможности пакета `DEtools()` могут отобразить страницу Справки командой `?DEtools`, на которой они найдут ссылки на страницы с описанием всех команд, входящих в пакет.

Они включают команды `DEplot()` и `DEplot3d()` для построения графиков решений систем дифференциальных уравнений, `PDEplot()` для решения уравнений в частных производных и его отображения (о ней пойдет речь в следующем разделе), `dfieldplot()` и `phaseportrait()` для отображения, соответственно, поля направлений и фазового портрета систем дифференциальных уравнений.

Команда `DEplot()` численно решает как одно обыкновенное дифференциальное уравнение любого порядка, так и нормальную систему обыкновенных дифференциальных уравнений, причем в этом случае должна быть только одна независимая переменная, т. е. переменная, от которой зависят искомые функции системы. Эта команда может использоваться с различным синтаксисом:

```
DEplot(deqns, vars, trange, inits, eqns);
DEplot(deqns, vars, trange, inits, xranges, eqns);
DEplot(deqns, vars, trange, xranges, eqns);
```

Параметр `deqns` задает либо одно дифференциальное уравнение произвольного порядка, либо систему в виде списка/множества, элементы которого представляют дифференциальные уравнения первого порядка, образующие систему.

Зависимые переменные системы или дифференциального уравнения, т. е. искомые функции, задаются параметром `vars`. В случае системы они должны быть представлены в виде списка/множества.

Параметром `trange` определяется диапазон изменения независимой переменной в виде:

```
t=a..b
```

В этом уравнении t задает имя используемой независимой переменной, а числовые параметры a и b определяют диапазон ее изменения. Еще раз напомним, что команда DEplot() работает с системой, в которой все неизвестные функции зависят от *одной* переменной.

Начальные условия определяются параметром inits, который представляет список, элементами которого являются списки (список списков). Каждый такой элемент-список определяет интегральную кривую дифференциального уравнения или системы, которая отображается на графике. Количество элементов-списков параметра inits соответствует количеству интегральных кривых на графике. Граничные условия задаются так же, как и для команды dsolve(), через дифференциальный оператор D. Например, следующий список определяет начальные условия для двух интегральных кривых одного дифференциального уравнения третьего порядка с неизвестной функцией z(x):

```
[[z(0)=1,D(z)(0)=2,(D$2)(z)(0)=0.5], [z(1)=1,D(z)(1)=2,(D$2)(z)(1)=0.5]]
```

Замечание

Если задаются граничные условия для отображения только одной интегральной кривой, то они также должны быть заданы как список:

```
[[z(0)=1,D(z)(0)=2,(D$2)(z)(0)=0.5]]
```

Замечание

Если для заданного дифференциального уравнения не существует интегральной кривой, удовлетворяющей некоторым заданным начальным условиям из списка, то команда не выдает никаких сообщений, отображая на результирующем графике только интегральные кривые, соответствующие действительным начальным условиям.

Параметры xranges, а их может быть столько, сколько неизвестных функций в системе, задают диапазоны изменения неизвестных функций и используются для завершения процесса интегрирования. Численное интегрирование осуществляется с заданным шагом, который по умолчанию равен abs(b-a)/20, где числа a и b задаются в параметре trange. Как только при очередном шаге значение какой-либо неизвестной функции выходит за пределы заданного в соответствующем параметре диапазона ее изменения, процесс интегрирования останавливается. Параметры xranges можно задавать в одной из двух форм:

```
x(t)=x1..x2
x=x1..x2
```

В них x представляет имя неизвестной функции дифференциального уравнения или системы, t — имя независимой переменной, а числа x1 и x2 задают, соответственно, нижнюю и верхнюю границу изменения неизвестной функции x(t).

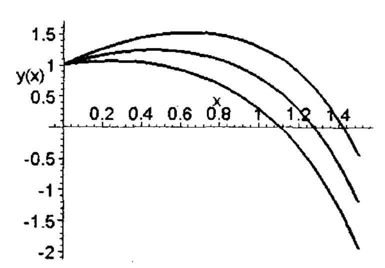
Необязательным параметром eqns задается ряд опций, определяющих общий вид графика решения: цвет линии интегральной кривой, шаг интегрирования, влияющий на гладкость отображения кривой, что и как откладывается по осям координат двумерного графика и т. д. Они, как и все опции в Maple, задаются в виде уравнений, в которых в левой части стоит имя опции, а в правой — ее значение. Кроме некоторых специальных опций они в основном совпадают с опциями команды plot() пакета plots, а также с опциями, которые можно задавать при построении численного решения (type=numeric) командой dsolve(). (Команда plot() описывается в гл. 4, посвященной графике в Maple.)

Пример 3.14. Отображение решений дифференциальных уравнений командой DEplot()

```
> with(DEtools):
> dif:=diff(y(x),x$2)-2*x*diff(y(x),x)+2*y(x)=0;
```

$$dif := \left(\frac{\partial^2}{\partial x^2}\mathrm{y}(x)\right) - 2\,x\left(\frac{\partial}{\partial x}\mathrm{y}(x)\right) + 2\,\mathrm{y}(x) = 0$$

```
> DEplot(dif,y(x),x=0..1.5,
      [[y(0)=1,D(y)(0)=0.5],[y(0)=1,D(y)(0)=1],[y(0)=1,D(y)(0)=1.5]],
      linecolor=black);
```




```
> # Задание уравнений системы дифференциальных уравнений
> dif1:=diff(x(t),t)=x(t)-y(t);
```

$$dif1 := \frac{\partial}{\partial t}\mathrm{x}(t) = \mathrm{x}(t) - \mathrm{y}(t)$$

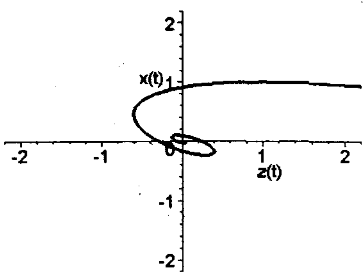
```
> dif2:=diff(y(t),t)=3*x(t)+y(t)-z(t);
```

$$dif2 := \frac{\partial}{\partial t}\mathrm{y}(t) = 3\,\mathrm{x}(t) + \mathrm{y}(t) - \mathrm{z}(t)$$

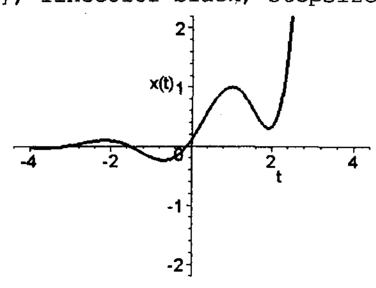
```
> dif3:=diff(z(t),t)=3*x(t)+y(t)+z(t);
```

$$dif3 := \frac{\partial}{\partial t}\mathrm{z}(t) = 3\,\mathrm{x}(t) + \mathrm{y}(t) + \mathrm{z}(t)$$

```
> # Отображение зависимости z(t) от x(t)
> DEplot({dif1,dif2,dif3},[x(t),y(t),z(t)],t=-4..4,
      [[x(1)=1,y(1)=1,z(1)=1]],x(t)=-2..2,y(t)=-2..2,z(t)=-2..2,
      scene=[z(t),x(t)], linecolor=black, stepsize=0.1);
```




```
> # Отображение зависимости x(t) от t
> DEplot({dif1,dif2,dif3},[x(t),y(t),z(t)],t=-4..4,
        [[x(1)=1,y(1)=1,z(1)=1]],x(t)=-2..2,y(t)=-2..2,z(t)=-2..2,
        scene=[t,x(t)], linecolor=black, stepsize=0.1);
```



В примере 3.14 решается дифференциальное уравнение второго порядка `dif` и отображаются три его интегральные кривые, проходящие через одну точку, но с разными углами наклона касательной (в начальных условиях для них задаются разные значения первой производной в точке `x=0`). Для системы трех дифференциальных уравнений первого порядка {`dif1`, `dif2`, `dif3`} строится график зависимости функции решения `z(t)` от функции `x(t)` и график решения `x(t)` относительно независимой переменной `t`. Это достигается использованием разных значений опции `scene`. Кроме нее для построения необходимых графиков в этом примере также используются опции `lynecolor` и `stepsize`, задающие, соответственно, цвет линии графика и шаг независимой переменной для получения точек графика. Команда `DEplot()` использует большой набор опций, наиболее употребительные из которых представлены в табл. 3.4.

Таблица 3.4. Опции команды `DEplot()`

Опция	Описание и значения
`arrows`	Задает размеры стрелок при отображении поля направлений для автономных линейных систем второго порядка или дифференциальных уравнений второго порядка. Допустимые значения: `SMALL` (установлен по умолчанию), `MEDIUM`, `LARGE`, `LINE`, `NONE`. Значение `NONE` подавляет отображение поля направлений

Таблица 3.4 (окончание)

Опция	Описание и значения
color	Задает цвет стрелок поля направлений
dirgrid	Определяет количество точек сетки в горизонтальном и вертикальном направлении для отображения векторов поля направлений. Значение задается в форме списка из двух целых чисел, первое из которых соответствует горизонтальному направлению, а второе вертикальному. Минимальное допустимое значение [2,2], максимальное [20,20], которое используется по умолчанию
iterations	Определяет количество шагов интегрирования между отображаемыми точками решения, значение по умолчанию равно единице. Данная опция полезна, когда для увеличения точности решения приходится уменьшать шаг интегрирования. Увеличение ее значения уменьшает объемы хранимой информации, так как в памяти хранятся только отображаемые точки решения
linecolor	Задает цвет выводимых линий решения. Если на одном графике отображается несколько решений дифференциального уравнения или системы, то можно задать список значений цветов, которыми будут отображены соответствующие решения. Если задан один цвет, то все решения на графике будут отображены этим цветом
obsrange	Установка этой опции, равной булевому значению TRUE (значение по умолчанию), останавливает процесс интегрирования системы дифференциальных уравнений, как только значение какой-либо искомой функции выходит за диапазон ее изменения, определенный параметром xrange. Значение FALSE отключает этот режим
scene	Определяет, что выводится на двумерном графике решения. Задается в виде двухэлементного списка искомых функций или независимой переменной. Например, scene=[x(t),y(t)] означает, что по горизонтальной оси графика отображается функция x(t), а по вертикальной — y(t). Это позволяет для системы второго порядка отобразить фазовый портрет. Для отображения решения следует задать эту опцию, например, в виде scene=[t,y(t)]
stepsize	Задает шаг изменения независимой переменной при численном интегрировании дифференциального уравнения. По умолчанию равен abs(b-a)/20, где [a,b] диапазон изменения независимой переменной, заданный параметром trange

Замечание

Остальные опции соответствуют опциям команды plot() и dsolve(), страницы справки которых можно отобразить командами ?plot[options] и ?dsolve[numeric].

Если система дифференциальных уравнений или одно дифференциальное уравнение, задаваемые в команде `DEplot()`, являются автономными, то задавать начальные значения не обязательно, однако обязательно задавать диапазон изменения искомых переменных. (Система дифференциальных уравнений называется автономной, если ее правые части не зависят явным образом от независимой переменной.) Для системы второго порядка по умолчанию рисуется поле направлений, и опция `arrows` определяет тип используемых стрелок, а опция `dirgrid` задает количество точек сетки для его представления. Если заданы начальные условия, то одновременно с полем направлений отображаются и эти решения. Для проверки автономности системы можно воспользоваться командой `autonomous()`, в качестве параметров которой передаются уравнения системы, искомые функции и независимая переменная. Эта команда возвращает истину, если система автономна.

Пример 3.15. Отображение решений и поля направлений автономной системы

```
> with(DEtools):
> dif1:=diff(x(t),t)=x(t)-y(t);
```

$$dif1 := \frac{\partial}{\partial t}\mathrm{x}(t) = \mathrm{x}(t) - \mathrm{y}(t)$$

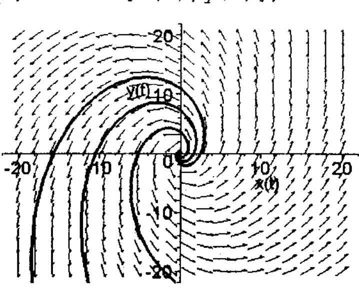
```
> dif2:=diff(y(t),t)=3*x(t)+y(t);
```

$$dif2 := \frac{\partial}{\partial t}\mathrm{y}(t) = 3\,\mathrm{x}(t) + \mathrm{y}(t)$$

```
> autonomous({dif1,dif2},[x(t),y(t)],t);
```

true

```
> DEplot({dif1,dif2},[x(t),y(t)],t=-4..4,
        [[x(1)=1,y(1)=1],[x(1)=2,y(1)=2],[x(1)=3,y(1)=3]],
        x(t)=-20..20,y(t)=-20..20,
        linecolor=black, scene=[x(t),y(t)], color=black, stepsize=0.1);
```



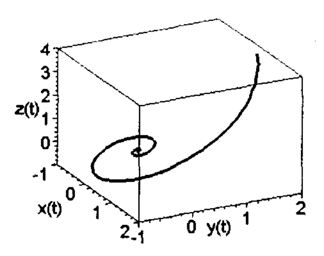
Команда `DEplot3d()` отображает в пространстве решения системы дифференциальных уравнений. Ее синтаксис аналогичен синтаксису команды `DEplot()`:

```
DEplot3d(deqns, vars, trange, inits, eqns);
Deplot3d(deqns, vars, trange, inits, xranges, eqns);
```

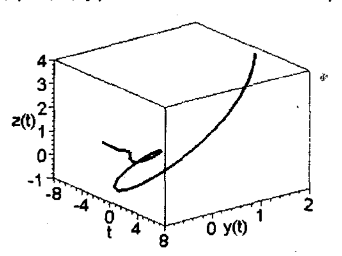
Смысл всех параметров аналогичен соответствующим параметрам команды DEplot(). Опции, задаваемые в виде уравнений опция=значение, такие же, как и перечисленные в табл. 3.4 для команды DEplot(), за исключением опций arrows, dirgrid и color, которые в ней не применяются, так как не строится, естественно, никакого поля направлений. Значения опции scene задаются в виде трехэлементного списка, комбинируемого из искомых функций и независимой переменной. Дополнительный набор опций соответствует опциям команды plot3d() из пакета plots и опциям команды dsolve при выполнении численного интегрирования. С ними можно познакомиться на страницах Справки, отображаемых, соответственно, командами ?plot3d[options] и ?dsolve[numeric].

Пример 3.16. Отображение в пространстве решений системы дифференциальных уравнений

```
> DEplot3d({dif1,dif2,dif3},[x(t),y(t),z(t)],t=-8..8,
        [[x(1)=1,y(1)=1,z(1)=1]],x(t)=-1..2,y(t)=-1..2,z(t)=-1..4,
        scene=[x(t),y(t),z(t)], linecolor=black, stepsize=0.1);
```




```
> DEplot3d({dif1,dif2,dif3},[x(t),y(t),z(t)],t=-8..8,
        [[x(1)=1,y(1)=1,z(1)=1]],x(t)=-1..2,y(t)=-1..2,z(t)=-1..4,
        scene=[t,y(t),z(t)], linecolor=black, stepsize=0.1);
```



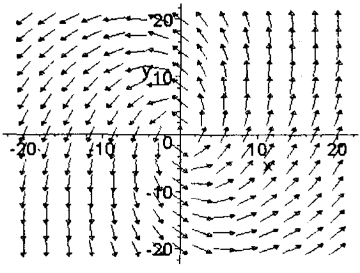
Для построения поля направлений и фазового портрета (совместно или без поля направлений) автономной системы двух линейных дифференциальных уравнений с одной независимой переменной можно воспользоваться командами dfieldplot() и phaseportrait() со следующим синтаксисом:

```
dfieldplot (deqns, vars, trange, xranges, eqns);
phaseportrait (deqns, vars, trange, inits, eqns);
```

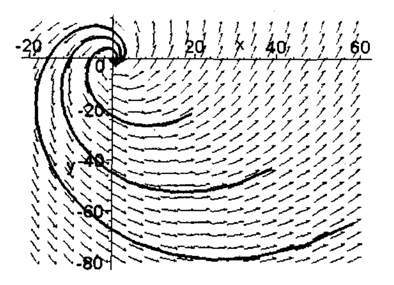
Для команды dfieldplot() опции, относящиеся к построению решений не действительны, а для команды phaseportrait() работают все опции команды DEplot(). Для системы дифференциальных уравнений из примера 3.15 построение этими командами поля направлений и фазового портрета демонстрируется в примере 3.17.

Пример 3.17. Поле направлений и фазовый портрет автономной системы

```
> dfieldplot({dif1,dif2},[x(t),y(t)],t=-4..4,
        x(t)=-20..20,y(t)=-20..20,
        color=black, arrows=MEDIUM, dirgrid=[15,15], stepsize=0.1);
```




```
> phaseportrait({dif1,dif2},[x(t),y(t)],t=-4..4,
        [[x(1)=1,y(1)=1],[x(1)=2,y(1)=2],[x(1)=3,y(1)=3]],
        linecolor=black, color=black, stepsize=0.1);
```



Замечание

Фактически команды dfieldplot() и phaseportrait() обращаются к команде DEplot().

3.4. Уравнения в частных производных

Задача интегрирования уравнений в частных производных достаточно сложная задача. Maple может находить решения некоторых уравнений в частных производных с помощью команды pdsolve(), которая находится в основной библиотеке. Она пытается найти общее аналитическое решение уравнения в

частных производных, передаваемое ей в качестве первого, и, возможно, единственного параметра. Это уравнение может быть произвольного типа (эллиптическое, гиперболическое, параболическое), любого порядка и с любым числом независимых переменных. Если обращение к команде `pdsolve()` не принесло результатов, то можно воспользоваться командами пакета `PDEtools` для преобразования уравнения в частных производных и приведения его к виду, решение которого снова попытаться найти командой `pdsolve()`. В этом же пакете есть команда `PDEplot()`, которая отображает численное решение уравнения в частных производных первого порядка.

3.4.1. Универсальная команда *pdsolve()*

Команда `pdsolve()` имеет две формы вызова:

```
pdsolve(pardif);
pdsolve(pardif,f,HINT=...,INTEGRATE,build);
```

Параметр `pardif` представляет уравнение в частных производных с одной неизвестной функцией. В версии Maple 6 пока нельзя решать системы уравнений в частных производных. Эта команда умеет находить общее решение для некоторых определенных типов уравнений в частных производных. Если ни один из известных ей методов не приводит к желаемому результату, то она включает эвристический алгоритм, который пытается осуществить разделение переменных с учетом структуры дифференциального уравнения в частных производных.

Общая стратегия данной команды заключается в нахождении общего решения, а если это не удается, то Maple пытается полностью разделить переменные. В случае успешного выполнения указанных задач, команда `pdsolve()` отображает результат в одном из следующих видов:

- общее решение;
- квази-общее решение, выраженное через некоторые произвольные функции, явный вид которых определяется соответствующими граничными и начальными условиями задачи;
- набор обыкновенных дифференциальных уравнений относительно всех разделенных переменных, или общее решение, если задан параметр `INTEGRATE`.

В первом случае решение представляется через произвольные функции, но их количество позволяет удовлетворить любым допустимым граничным и начальным условиям. Во втором случае, когда количество произвольных функций не может обеспечить решение задачи с произвольными граничными и начальными условиями, используется введенная в Maple 6 специальная структура `PDESolStruc` представления квази-решения, в которой перечисля-

ются разделенные переменные, а после ключевого слова &where отображается множество обыкновенных дифференциальных уравнений, которым удовлетворяют разделенные переменные.

Во второй форме вызова команды pdsolve() параметры имеют следующий смысл. После дифференциального уравнения следует задать функцию, относительно которой задано дифференциальное уравнение, если уравнение содержит производные более чем одной неизвестной функции. Остальные параметры являются необязательными. Можно построить явный вид решения вне зависимости от того, получено ли общее или квази-общее решение, указав опцию `build`. В случае квази-решения с разделенными переменными опция INTEGRATE приводит к немедленному интегрированию полученных обыкновенных дифференциальных уравнений из структуры PDESolStruc.

Последний оставшийся параметр HINT используется для указания, с какого метода разделения переменных следует начать поиск решения. Он задается в виде уравнения, правая часть которого и определяет метод разделения. Допустимыми значениями могут быть `+` для задания разделения переменных в виде суммы, `*` в виде произведения или любое выражение, содержащее некоторые неопределенные функции, зависящие от независимых переменных функций дифференциального уравнения и указывающие, каким образом следует представить решение уравнения и разделить переменные. Например, для одномерного волнового уравнения можно использовать метод Фурье разделения переменных, представляя решение в форме произведения двух функций, первая из которых зависит только от одной независимой переменной, а вторая только от второй. В этом случае можно задать следующую опцию HINT=X(x)*T(t), где x и t — независимые переменные задачи. Последнее допустимое значение этой опции `strip` применяется к уравнениям первого порядка и определяет использование метода характеристик для получения общего решения.

Пример 3.18. Решение уравнений в частных производных

```
> wave:=diff(u(x,t),t$2)=a^2*diff(u(x,t),x$2);
```

$$wave := \frac{\partial^2}{\partial t^2}\mathrm{u}(x,t) = a^2\left(\frac{\partial^2}{\partial x^2}\mathrm{u}(x,t)\right)$$

```
> # Общее решение волнового уравнения
> pdsolve(wave);
```

$$\mathrm{u}(x,t) = _F1(a\,t+x) + _F2(a\,t-x)$$

```
> # Квази-общее решение волнового уравнения методом разделения переменных
> pdsolve(wave,HINT=X(x)*T(t));
```

$$(\mathrm{u}(x,t) = \mathrm{X}(x)\,\mathrm{T}(t))\ \&\mathrm{where}\left[\ \{\frac{\partial^2}{\partial x^2}\mathrm{X}(x) = _c_1\,\mathrm{X}(x), \frac{\partial^2}{\partial t^2}\mathrm{T}(t) = a^2\,_c_1\,\mathrm{T}(t)\}\ \right]$$

```
> # Квази-общее решение волнового уравнения методом разделения переменных
> # с интегрированием полученных обыкновенных дифференциальных уравнений
> pdsolve(wave,HINT=X(x)*T(t),INTEGRATE);
```

$$(u(x,t)=X(x)\,T(t))\ \&\text{where}$$

$$[\{\{T(t)=_C3\,e^{(\sqrt{_c_1}\,a\,t)}+_C4\,e^{(-\sqrt{_c_1}\,a\,t)}\},\{X(x)=_C1\,e^{(\sqrt{_c_1}\,x)}+_C2\,e^{(-\sqrt{_c_1}\,x)}\}\}]$$

```
> # Построение решения в явной форме для квази-общего решения волнового
> # уравнения
> pdsolve(wave,HINT=X(x)*T(t),build);
```

$$u(x,t)=$$

$$e^{(\sqrt{_c_1}\,x)}\,_C3\,_C1\,e^{(\sqrt{_c_1}\,a\,t)}+\frac{e^{(\sqrt{_c_1}\,a\,t)}\,_C3\,_C2}{e^{(\sqrt{_c_1}\,x)}}+\frac{_C4\,_C1\,e^{(\sqrt{_c_1}\,x)}}{e^{(\sqrt{_c_1}\,a\,t)}}+\frac{_C4\,_C2}{e^{(\sqrt{_c_1}\,x)}\,e^{(\sqrt{_c_1}\,a\,t)}}$$

Общее решение получено через две произвольные функции `_F1` и `_F2`. Команда `pdsolve()` всегда использует префикс `_F` с последующим целым числом для представления произвольных функций общего решения уравнений в частных производных. Обратите также внимание на различие в ответах при использовании параметра `INTEGRATE` и без такового. В первом случае выводится множество обыкновенных дифференциальных уравнений с произвольным параметром-константой $_c_1$, тогда как во втором случае отображаются общие решения этих обыкновенных дифференциальных уравнений с произвольными константами *_C1*, *_C2*, *_C3* и *_C4*. При использовании опции `build` получаемое решение зависит от четырех произвольных констант и одного упоминавшегося параметра-константы.

После построения общего решения можно отобразить частное решение функцией `plot3d()` из пакета `plots`. Покажем, как можно это сделать для общего решения одномерного волнового уравнения:

```
> wave:=diff(u(x,t),t$2)=a^2*diff(u(x,t),x$2);
```

$$wave:=\frac{\partial^2}{\partial t^2}u(x,t)=a^2\left(\frac{\partial^2}{\partial x^2}u(x,t)\right)$$

```
> sol:=pdsolve(wave);
```

$$sol:=u(x,t)=_F1(a\,t+x)+_F2(a\,t-x)$$

Сохранив решение в переменной `sol`, нам следует задаться явным видом произвольных функций общего решения `_F1` и `_F2`, а также параметру `a` уравнения придать конкретное значение. Это можно осуществить следующими командами:

```
> f1:=x->exp(-x^2);
```

$$f1:=x\to e^{(-x^2)}$$

```
> f2:=x->piecewise(x>-1/2 and x<1/2,1,0);
```

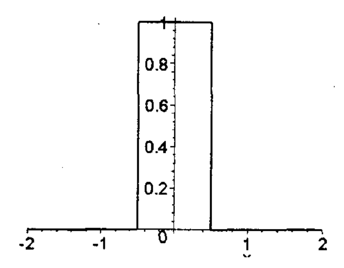
$$f2 := x \to \text{piecewise}\left(\frac{-1}{2} < x \text{ and } x < \frac{1}{2}, 1, 0\right)$$

```
> a:=1;
```

$$a := 1$$

Здесь сделаем небольшое отступление в связи с заданием функции командой `piecewise()`. Эта команда задает кусочно-непрерывную функцию. Ее параметры задаются парами: первый определяет интервал изменения независимой переменной, а второй значение функции на этом интервале. Последний непарный параметр определяет значение функции в оставшихся диапазонах изменения независимой переменной. Кусочно-непрерывные функции можно дифференцировать, интегрировать, отображать их графики. Вообще, с ними можно обращаться как с обычными непрерывными функциями: система Maple очень хорошо умеет с ними работать. Например, график функции `f2(x)` строится обычной командой `plot()`:

```
> plot(f2(x),x=-2..2,color=black,thickness=2);
```



Теперь остается подставить введенные функции в полученное общее решение sol, выделить правую часть, преобразовать ее в функцию и отобразить:

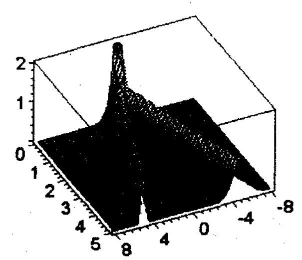
```
> eval(sol,{_F1=f1,_F2=f2});
```

$$u(x,t) = e^{(-(t+x)^2)} + \left(\begin{cases} 1 & -t+x < \frac{1}{2} \text{ and } t-x < \frac{1}{2} \\ 0 & \textit{otherwise} \end{cases}\right)$$

```
> f:=unapply(rhs(%),x,t);
```

$$f := (x,t) \to e^{(-(t+x)^2)} + \text{piecewise}\left(-t+x < \frac{1}{2} \text{ and } t-x < \frac{1}{2}, 1, 0\right)$$

```
> plot3d(f,-8..8,0..5,grid=[40,40]);
```



Обратите внимание, как Maple отображает кусочно-непрерывную функцию, когда она используется в выражении.

Замечание

Рисунок решения, получаемый функцией `plot3d()` с заданными в тексте значениями параметров, не будет содержать оси координат и будет цветным. Рисунок, приведенный в книге, был изменен вручную с помощью команд форматирования контекстной панели инструментов.

3.4.2. Команды пакета *PDEtools*

Как и в случае с обыкновенными дифференциальными уравнениями, если не удается построить решение уравнения в частных производных командой `pdsolve()`, то можно воспользоваться командами пакета `PDEtools` для преобразования уравнения с целью попытаться свести его к уравнению, которые можно решить этой командой. В табл. 3.5 представлены все команды пакета `PDEtools` с их кратким описанием.

Таблица 3.5. *Команды пакета* `PDEtools`

Команда	Описание
`build`	Решает обыкновенные дифференциальные уравнения, получаемые при использовании метода разделения переменных в команде `pdsolve()`, и возвращает явный вид решения уравнения в частных производных. Используется командой `pdsolve()` с заданной опцией `build`
`dcoeffs`	Приводит подобные коэффициенты в полиномиальном уравнении в частных производных относительно неизвестной и степеней ее производных и выдает список полученных коэффициентов
`dsubs`	Входными параметрами команды являются список уравнений, в левых частях которых стоят производные от неизвестных функций, входящих в выражение, представленное самым последним параметром. Команда последовательно подставляет в это выражение соответствующие производные, причем пытается продолжить подстановку до тех пор, пока из выражения не будут исключены все производные, определяемые в передаваемых команде уравнениях
`casesplit`	Разбивает систему уравнений и/или неравенств на последовательность систем уравнений и неравенств таких, что объединение их не особых решений равняется множеству решений исходной системы. Кроме того, каждая из полученных систем свободна от дифференциальной или алгебраической избыточности, и для них автоматически удовлетворены условия интегрируемости

Таблица 3.5 (окончание)

Команда	Описание
charstrip	Вычисляет уравнение характеристик для заданного дифференциального уравнения в частных производных первого порядка, т. е. строится и решается соответствующая ему система обыкновенных дифференциальных уравнений в симметрической форме
dchange	Осуществляет замену переменных в любых алгебраических объектах (уравнения в частных производных, кратных интегралах, интегродифференциальных уравнениях и т. д.), а также в процедурах. Эта команда полезна для изменения уравнения в частных производных от формы, трудно решаемой, к форме, для которой можно найти решение
difforder	Определяет общий или относительно некоторой переменной порядок частной производной в уравнении или выражении
mapde	Преобразует уравнение в частных производных к уравнению в частных производных другого вида, которое, возможно, будет решено командой pdsolve()
PDEplot	Строит график решения уравнения в частных производных первого порядка, линейного или нелинейного, при заданных начальных условиях
pdetest	Проверяет, является некоторое выражение решением уравнения в частных производных, возвращая 0 в случае положительности проверки и выражение невязки в противном случае
separability	Определяет, при каких условиях для заданного уравнения в частных производных возможно получение полного решения при разделении переменных в виде суммы или произведения
splitsys	Разбивает систему алгебраических или дифференциальных уравнений на независимые между собой системы уравнений
splitstrip	Вычисляет характеристики уравнения в частных производных, но возвращает их, если это возможно, разбитыми на подмножества характеристик независимых между собой систем уравнений, полученных в результате выполнения команды splitsys()

Как и в случае с пакетом DEtools мы опишем команду PDEplot(), так как остальные команды пакета DEtools() скорее всего окажутся полезными для специалиста в области уравнений в частных производных, чем для рядового технического работника.

Общий синтаксис команды PDEplot() следующий:

```
PDEplot(PDE, inits, srange, options);
```

Здесь параметром PDE задается уравнение в частных производных первого порядка относительно одной неизвестной функции, зависящей от n переменных. Начальные условия определяются параметром inits в форме списка из $n+1$ элементов, определяющих в параметрической форме кривую в пространстве $n+1$ измерений, через которую проходит интегральная поверхность дифференциального уравнения. Элементы должны быть выражениями, зависящими от $n-1$ параметра. Параметром srange задаются диапазоны изменения каждого параметра, используемого в начальных условиях в виде s=s1..s2, t=t1..t2, ... Параметр options задает опции в виде уравнений, в которых левая часть представляет имя опции, а правая — его значение. Смысл некоторых опций совпадает с аналогичными, используемыми в команде DEplot(). Перечень всех опций команды PDEplot() с кратким описанием представлен в табл. 3.6.

Таблица 3.6. *Опции команды* PDEplot()

Опция	Описание и значения
iterations	Определяет количество шагов интегрирования между отображаемыми точками решения, умалчиваемое значение равно единице. Данная опция полезна, когда для увеличения точности решения приходится уменьшать шаг интегрирования. Увеличение ее значения уменьшает объемы хранимой информации, так как в памяти хранятся только отображаемые точки решения
stepsize	Задает расстояние между вычисляемыми точками вдоль каждой из характеристик, которое может принимать вещественные значения. Значение по умолчанию, оно же и максимальное, равняется 0.25
numsteps	Определяет количество отображаемых точек вдоль характеристической линии в каждом направлении. Ее значения в этом случае задаются в виде двухэлементного списка целых чисел со знаком. Знак определяет направление. Если задано просто целое со знаком, то характеристическая кривая отображается только в соответствующем направлении, определяемом знаком числа. По умолчанию эта опция имеет значение [-10,10]
numchar	Гиперповерхность решения строится из характеристических линий уравнения, каждая из которых проходит через определенную точку начального многообразия. Эта опция определяет количество таких точек, задавая дискретное изменение каждого параметра, используемого в описании многообразия начальных значений. Задается в форме списка или одного целого, которое относится ко всем параметрам. Значение по умолчанию равно 20

Таблица 3.6 *(продолжение)*

Опция	Описание и значения
scene	Определяет, что выводится на трехмерном графике решения. Задается в виде трехэлементного списка искомых функций или независимых переменных. Например, scene= =[x1,x2,u(x1,x2)] означает, что по оси x и y графика отображается независимые переменные x1 и x2, а по вертикальной оси z функция решения (это значение по умолчанию)
xi=ximin..ximax, u(x1,x2,...,xn) = u_min..u_max	Можно определять диапазон изменения отображаемых величин, указанных в опции scene. Если какая-то из отображаемых величин вышла за пределы заданного для нее диапазона изменения, то процесс отображения решения на этом прекращается
obsrange	Установка этой опции, равной булевому значению TRUE (значение по умолчанию), останавливает процесс интегрирования вдоль характеристической кривой решения дифференциального уравнения, как только значение какой-либо отображаемой переменной выходит за заданный для нее диапазон изменения. Значение FALSE отменяет это действие
method	Определяет численный метод интегрирования вдоль характеристик. По умолчанию используется внутренний метод Рунге-Кутта. Можно устанавливать любой метод, который используется в команде dsolve() при численном интегрировании обыкновенных дифференциальных уравнений, но следует учитывать, что использование метода, отличного от внутреннего, приводит к существенному увеличению времени расчета
animate	В общем случае дифференциального уравнения, в котором неизвестная функция зависит от *n* переменных, решение представляет собой n-мерную гиперповерхность в (*n*+1)-мерном пространстве. Для *n*>2 ее, естественно, сложно отобразить в трехмерном пространстве. Можно создать анимационную картинку, отображающую последовательность многообразий, которые все вместе составляют поверхность решения. Для этого следует установить значение опции animate, равной true. При значении опции animate=false отображается гиперповерхность решения с выделенным черным цветом начальным условием. Если animate=only, то поверхность решения никак не отображается, а вместо нее отображается последовательность начальных многообразий, из которой можно составить представление о поверхности решения. По умолчанию animate=true при *n*=2 и animate=false при *n*>2

Таблица 3.6 (окончание)

Опция	Описание и значения
`ic_assumptions`	Определяет список возможных начальных условий для первых производных неизвестных функций в случае нелинейного уравнения первого порядка в частных производных
`basechar`	Значение этой опции определяет, будут ли отображаться базовые кривые характеристик (проекции начальных условий на плоскость `x-y`). Если ее значение равно `true`, то отображаются, если равно `false`, то не отображаются. При значении, равном `only`, будут отображаться только базовые характеристики и кривая начальных данных — поверхность решения отображаться не будет. Значение по умолчанию — `false`
`color`	Определяет используемый для закрашивания поверхности цвет
`initcolor`	Определяет используемый для закрашивания гиперповерхностей начальных условий цвет

Замечание

Остальные опции соответствуют опциям команды `plot3d()` (кроме `grid`, `gridstyle` и `numpoints`) и `dsolve[numeric]`, страницы справки которых можно отобразить командами `?plot3d[options]` и `?dsolve[numeric]`.

Теперь мы применим команду `PDEplot()` для решения уравнения

$$y\,z(x,y)\left(\frac{\partial}{\partial x}z(x,y)\right)+x\left(\frac{\partial}{\partial y}z(x,y)\right)=0$$

со следующими начальными условиями: $z=x^2$ при $y=1$.

Пример 3.19. Отображение решения линейного уравнения в частных производных

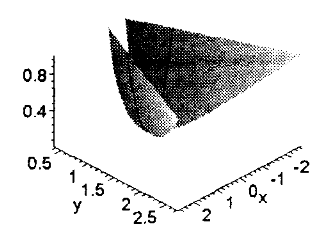
```
> eq:=y*z(x,y)*diff(z(x,y),x)+x*diff(z(x,y),y)=0;
```

$$eq:=y\,z(x,y)\left(\frac{\partial}{\partial x}z(x,y)\right)+x\left(\frac{\partial}{\partial y}z(x,y)\right)=0$$

```
> inits:=[s,1,s^2];
```

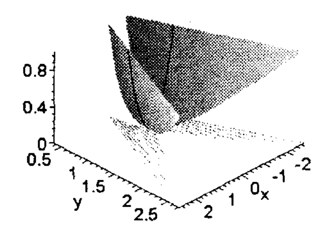
$$inits:=[s,1,s^2]$$

```
> PDEplot(eq, inits, s=-1..1, stepsize=0.1);
```



Начальные условия представляют собой гиперповерхность размерности 1 в пространстве решения размерности 3 (количество независимых переменных n=2), т. е. обычную кривую в пространстве, которая в нашем примере является к тому же плоской кривой. Для ее задания в команде `PDEplot()` мы определяем список, состоящий из трех ее координат, зависящих от параметра `s`, за который мы приняли независимую переменную `x`. Обратите внимание на то, что кривая начальных условий также отображается на графике. Теперь дополнительно отобразим вместе с решением базовые характеристики:

```
> PDEplot(eq,inits,s=-1..1,stepsize=0.1,basechar=true);
```



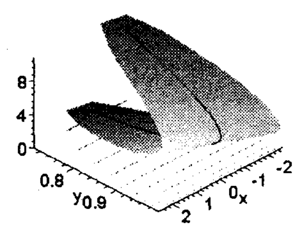
В заключение этого раздела приведем пример решения нелинейного уравнения в частных производных первого порядка.

Пример 3.20. Отображение решения нелинейного уравнения в частных производных

```
> pde2 := diff(u(x,y),x)^diff(u(x,y),x)=y;
```

$$pde2 := \left(\frac{\partial}{\partial x}\mathrm{u}(x,y)\right)^{\left(\frac{\partial}{\partial x}\mathrm{u}(x,y)\right)} = y$$

```
> PDEplot(pde2,[cos(t),sin(t),exp(t)],Pi/4..3*Pi/4,
        animate=false, basechar=true);
```



В примере 3.20 задается параметр `animate=false`, чтобы не создавать анимационную картинку, которая была бы построена по умолчанию. Начальные условия задаются непосредственно при обращении к команде `PDEplot()`.

3.5. Другие пакеты

Завершая разговор о пакетах Maple, в которых сосредоточена не малая функциональность этой системы аналитических вычислений, последний раздел мы посвятили краткому описанию некоторых полезных, с нашей точки зрения, пакетов.

3.5.1. Пакет *student*

Пакет `student` разрабатывался специально для студентов, чтобы предоставить набор команд для пошаговой реализации основополагающих математических методов, изучаемых в курсе высшей математики. Он позволяет приближать интегралы правыми и левыми суммами, производить вычисление интегралов заменой переменной и интегрированием по частям, получать приближенное значение определенных интегралов методами прямоугольников, трапеций и Симпсона, находить максимальные значения функции на заданном интервале и т. д. Команды этого пакета освобождают студента от выполнения некоторых "механических" операций, например вычисления производных, и концентрирует его внимание на правильном применении необходимых правил вычисления математических величин, например интегралов. В него входят следующие команды:

```
D, Diff, Doubleint, Int, Limit, Lineint, Point, Product, Sum, Tripleint,
changevar, combine, completesquare, distance, equate, extrema, integrand,
intercept, intparts, isolate, leftbox, leftsum, makeproc, maximize,
middlebox, middlesum, midpoint, minimize, powsubs, rightbox, rightsum,
showtangent, simpson, slope, summand, trapezoid
```

Мы не будем рассматривать каждую команду этого пакета: их назначение, как нам кажется, следует из их имен, и более того, в любой момент пользователь может получить исчерпывающую информацию обо всех этих командах с многочисленными примерами их использования, обратившись к справочной системе Maple, но несколько примеров дадут представление об этом пакете. В части II книги, посвященной применению Maple для решения математических задач, мы интенсивно будем использовать команды этого пакета, где многие из них и будут описаны.

Пусть необходимо найти значение следующего определенного интеграла

$$\int_0^1 \frac{3}{4}x - x^2\,dx$$

на основе его определения через центральную сумму.

Прежде всего определим подынтегральную функцию и построим для интеграла от нее на заданном интервале центральную сумму, используя `n` центральных прямоугольников:

```
> f:=x->3/4*x-x^2;
```

$$f := x \to \frac{3}{4}x - x^2$$

```
> midsum:=middlesum(f(x),x=0..1,n);
```

$$midsum := \frac{\sum_{i=0}^{n-1}\left(\frac{3}{4}\,\frac{i+\frac{1}{2}}{n} - \frac{\left(i+\frac{1}{2}\right)^2}{n^2}\right)}{n}$$

Эту сумму можно вычислять при различных значениях параметра n, получая приближенное значение интеграла. Для точного значения необходимо вычислить ее предел при $n \to \infty$:

```
> Limit(midsum,n=infinity);
```

$$\lim_{n \to \infty} \frac{\sum_{i=0}^{n-1}\left(\frac{3}{4}\,\frac{i+\frac{1}{2}}{n} - \frac{\left(i+\frac{1}{2}\right)^2}{n^2}\right)}{n}$$

```
> value(%);
```

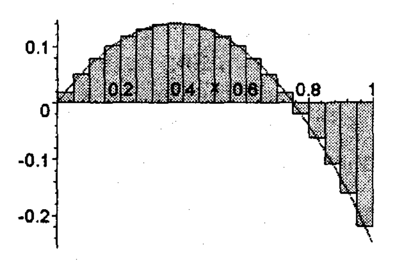
$$\frac{1}{24}$$

Теперь можно проверить полученный результат командой `int()` и даже построить приближение центральными прямоугольниками искомого интеграла:

```
> int(f(x),x=0..1);
```

$$\frac{1}{24}$$

```
> middlebox(f(x),x=0..1,20);
```



3.5.2. Связь с Matlab

Пакет `Matlab` позволяет непосредственно из сеанса Maple обращаться к ограниченному множеству функций популярного среди инженеров пакета численных вычислений Matlab при условии, что он установлен на компьютере пользователя. При подключении команд пакета, если он не установлен, отображается соответствующее предупреждение:

```
> with(Matlab);
Error, (in execSystemInit) system level initialisation for package
      `Matlab' failed
```

При успешном подключении пакета в области вывода печатаются, как обычно, все доступные команды пакета:

```
> with(Matlab);
```

chol, closelink, defined, det, dimensions, eig, evalM, fft, getvar
inv, lu, ode45, openlink, qr, setup, setvar, size, square, transpose

Чтобы установить связь между двумя программами (Maple и Matlab), следует прежде всего выполнить команду `openlink()` без каких-либо параметров. После этого можно использовать все команды пакета для выполнения необходимых действий. Завершается сеанс связи выполнением команды `closelink()`.

Функции Matlab могут работать как с объектами Maple (массивы, матрицы и векторы на основе таблиц, массивы, матрицы и векторы на основе r-таблиц, вещественные с плавающей точкой и целые числа), так и с объектами Matlab. Более подробную информацию о работе с пакетом `Matlab` можно получить в справочной системе, выполнив команду `?Matlab`.

Внимание!

В одном сеансе Maple можно открыть только один сеанс связи с Matlab. Поэтому все переменные Matlab, определенные в Maple, доступны для всех рабочих листов текущего сеанса Maple, даже если Maple работает в параллельном режиме, т. е. переменные одного рабочего листа Maple не видимы для другого.

3.5.3. Пакет линейной оптимизации simplex

В практике математического моделирования линейная оптимизационная модель занимает одно из важных мест, так как достаточно большое количество оптимизационных задач, возникающих в реальной жизни, хорошо описываются этой моделью. В основном эти задачи возникают из экономической деятельности человека, хотя это и не обязательно.

Задача линейной оптимизации формулируется в следующем виде: минимизировать линейную функцию нескольких аргументов при ограничениях в

виде равенств и неравенств на независимые переменные. Для ее решения применяется симплекс-метод, теория которого хорошо разработана.

В Maple команды пакета `simplex` как раз и реализуют симплекс-метод для решения задач линейной оптимизации, которые еще называют задачами линейного программирования. Набор команд позволяет непосредственно решить задачу линейного программирования за одно обращение к функции `maximize()` или `minimize()`, или по шагам строить решение на основе алгоритма симплекс-метода, или просто исследовать задачу линейного программирования, например, определить, допустимы ли ограничения, построить двойственную задачу и т. п.

В качестве примера решим задачу нахождения максимального значения функции

$$l(x, y, z) = -x + 2\,y + 3\,z$$

при следующих ограничениях:

$$x + 2\,y - 3\,z \le 4$$
$$5\,x - 6\,y + 7\,z \le 8$$
$$9\,x + 10\,z \le 11$$

Кроме этих ограничений будем предполагать, что все независимые переменные неотрицательны. Решение с помощью пакета `simplex` представлено ниже:

```
> l:=-x+2*y+3*z;
```

$$l := -x + 2\,y + 3\,z$$

```
> cns1:=x+2*y-3*z<=4;
```

$$cns1 := x + 2\,y - 3\,z \le 4$$

```
> cns2:=5*x-6*y+7*z<=8;
```

$$cns2 := 5\,x - 6\,y + 7\,z \le 8$$

```
> cns3:=9*x+10*z<=11;
```

$$cns3 := 9\,x + 10\,z \le 11$$

```
> with(simplex);
Warning, the protected names maximize and minimize have been redefined
and unprotected
```

[*basis*, *convexhull*, *cterm*, *define_zero*, *display*, *dual*, *feasible*, *maximize*, *minimize* , *pivot* , *pivoteqn* , *pivotvar*, *ratio*, *setup*, *standardize*]

```
> maximize(l,{cns1,cns2,cns3},NONNEGATIVE);
```

$$\{z = \frac{11}{10}, y = \frac{73}{20}, x = 0\}$$

Обратите внимание, что при подключении пакета `simplex` отображается предупреждение, что команды `maximize()` и `minimize()` были переопределе-

ны. Это означает, что теперь эти команды соответствуют командам пакета simplex, а не командам с такими же именами из основной библиотеки. Чтобы снова их использовать, можно воспользоваться командой restart, которая, правда, также отменяет все осуществленные присваивания переменным в текущем сеансе Maple.

3.5.4. Пакет статистики stats

Пакет статистик stats содержит огромное количество команд для обработки, анализа и отображения статистических данных. Также он содержит большое количество статистических распределений.

Этот пакет является примером пакета, состоящим из подпакетов, в которых сгруппированы команды, относящиеся к определенным разделам статистики. Например, подпакет describe содержит все необходимые команды для анализа данных, команды подпакета random позволяют получить любое известное статистическое распределение в виде списка данных и т. д. Всего пакет stats содержит семь подпакетов и одну функцию importdata(), позволяющую импортировать данные из внешнего файла. Можно подключить все команды пакета с помощью

```
> with(stats);
```

или команды отдельных пакетов

```
> with(stats, имя_подпакета);
```

Команды подпакетов вызываются с использованием полных имен:

```
> имя_подпакета[имя_команды](параметры);
```

Для использования коротких имен команд подпакета следует после подключения всего пакета или отдельного его подпакета выполнить команду:

```
> with(имя_подпакета);
```

Команды пакета stats работают с данными, расположенными в статистических списках, которые включают в себя и обычные списки Maple. Специальные статистические списки могут включать в себя диапазоны и веса (количество повторений в списке заданной величины):

```
> restart:with(stats);
```

$$[anova, describe, fit, importdata, random, statevalf, statplots, transform]$$

```
> statlist:=[Weight(0.7,3),Weight(1.07,4),0.945,
        Weight(0.02..0.03,5),0.896,1.01,1.04];
```

$$statlist := [\mathrm{Weight}(.7, 3), \mathrm{Weight}(1.07, 4), .945, \mathrm{Weight}(.02 .. .03, 5), .896, 1.01, 1.04]$$

Выражение Weight(x,n) указывает, что величина x появляется в списке n раз. Если конкретные значения из какого-то диапазона не являются значимыми и их можно не различать, то для таких случаев предусмотрено задание диапазона.

После создания статистического списка можно вычислить его характеристики командами подпакета describe:

```
> describe[mean](statlist); # Статистическое среднее
```

$$\mathrm{mean}(\mathit{statlist})$$

```
> describe[standarddeviation](statlist); # Стандартное отклонение
```

$$.4398483402$$

Получить список значений из нормального распределения можно командой normald подпакета random:

```
> normal_data:=[random[normald](30)];
```

normal_data :=[-.9793892252, .5858612336, 1.253735483, -.6332354069, .9726524838, -.5598900697, .6789450618, -.2075898850, .5229930444, 2.534480124, .5533219809, -2.463631304, .2949888635, .8577412317, -1.258903656, .6595246058, 1.196309627, .2113765841, -1.297600422, -1.369469750, -.4141431298, -.3832449896, -.3953803072, -.9252182176, .5623788414, 1.569545551, -1.456403600, .3170756338, -1.312018094, -.03770603101]

Теперь можно подсчитать, сколько значений из полученного списка попадают в определенные интервалы и построить гистограмму для этих данных, чтобы визуально оценить распределение, которому они принадлежат:

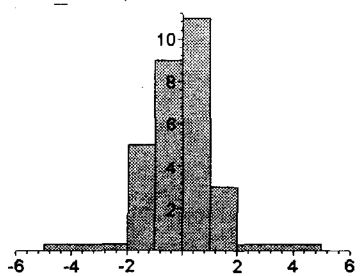
```
> ranges:=[-5..-2,-2..-1,-1..0,0..1,1..2,2..5];
```

$$\mathit{ranges} := [-5 \,..\, -2, -2 \,..\, -1, -1 \,..\, 0, 0 \,..\, 1, 1 \,..\, 2, 2 \,..\, 5]$$

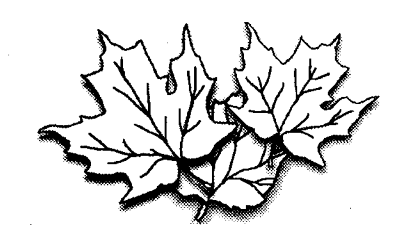
```
> data_list:=transform[tallyinto](normal_data,ranges);
```

data_list :=[Weight(-1 .. 0, 9),Weight(0 .. 1, 11),Weight(1 .. 2, 3), 2 .. 5, -5 .. -2, Weight (-2 .. -1, 5)]

```
> statplots[histogram](data_list);
```



ГЛАВА 4



Графика

Системы аналитических вычислений привлекают исследователей не только своими возможностями реализации алгоритмов построения аналитических решений, но и развитой графикой, начиная от построения простейших двумерных кривых и заканчивая сложными трехмерными поверхностями и анимацией двумерных и трехмерных изображений. В любой момент пользователь может отобразить результаты своих вычислений в виде графических образов, которые, как известно, более информативны, чем скупые ряды цифр. (Хотя, справедливости ради, следует заметить, что в некоторых ситуациях цифра может оказаться более полезной, чем общая картина изменения какого-либо параметра.)

Универсальные графические команды собраны в пакете `plots`, а в подпакете `statplots` пакета `stats` находятся специальные команды отображения статистических данных. Команды построения графиков численных решений обыкновенных дифференциальных уравнений `DEplot()` и уравнений в частных производных `PDEplot()` можно найти, соответственно, в пакетах `DEtools` и `PDEtools`. Пакет `student` содержит несколько иллюстративных команд представления определенных интегралов в виде различных сумм, а также команду отображения касательной к функции в заданной точке. Чтобы воспользоваться перечисленными графическими средствами, обязательно подключение соответствующих пакетов. Но в Maple есть две всегда доступные графические команды `plot()` и `plot3d()`, которые расположены в основной библиотеке. Первая предназначена для построения графиков функций одной переменной (двумерная графика); с помощью второй можно строить трехмерные графические отображения поверхностей и пространственных кривых (пространственная графика). Команды в указанных выше пакетах также можно подразделить на команды двумерной и пространственной графики.

4.1. Команды двумерной графики

4.1.1. Команда *plot()*

Многофункциональная двумерная графическая команда plot() расположена в системной библиотеке Maple, и поэтому доступна в любое время. Именно с нее мы и начнем знакомство с графическими возможностями системы аналитических вычислений Maple.

С помощью этой команды можно построить график одной или нескольких функций одной вещественной переменной, заданных в явном или параметрическом виде, а также отобразить множество точек в декартовой или полярной системе координат. Синтаксис команды plot() следующий:

```
plot(f, h, v, опции);
```

Здесь f — функция, график которой необходимо отобразить, h и v представляют, соответственно, диапазон изменения независимой переменной по горизонтальной оси графика и диапазон изменения значения функции вдоль вертикальной оси графика.

Диапазон изменения независимой переменной h задается в виде x=a..b, где a и b — наименьшее и наибольшее значения изменения переменной, а x — имя независимой переменной. Если диапазон не задан, т. е. второй параметр представляет собой просто имя независимой переменной в функции, то по умолчанию принимается следующий интервал ее изменения -10..10. Этот параметр (с диапазоном или нет) обязательно должен присутствовать при задании графика командой plot().

Вертикальный диапазон v, задаваемый третьим параметром, ограничивает вывод графика определенной областью изменения функции. Он необязателен, как и опции, задающиеся в виде уравнений имя_опции=значение. При отсутствии явного задания опций принимаются их значения по умолчанию.

Опции определяют вид отображаемого графика: толщину, цвет и тип линии графика, тип осей координат, размещение надписей и т. д. и задаются в форме уравнений имя_опции=значение. Набор возможных опций во всех командах двумерного графического вывода, за некоторым исключением, одинаков. В табл. 4.1 представлены все опции двумерной графики и соответствующие им значения (умалчиваемые значения подчеркнуты).

***Таблица 4.1.** Опции двумерной графики*

Опция	**Описание**
adaptive	Maple использует специальный адаптивный алгоритм для вычисления отображаемых точек кривой: сначала вычисляются значения функции на некотором множестве равноотстоящих

Таблица 4.1 *(продолжение)*

Опция	Описание
(прод.)	точек в заданном интервале изменения независимой переменной, а затем в областях, где график функции сильно отличается от прямой линии, соединяющей соседние точки, вычисляются значения функции в дополнительных точках. По умолчанию этот алгоритм всегда включен (значение опции равно true), но его можно отключить, установив значение опции adaptive равным false
axes	Определяет тип отображаемых осей координат. Эта опция может принимать следующие значения: NORMAL — обычные оси координат, пересекающиеся в точке начала координат (0,0); BOXED — график заключен в прямоугольник с нанесенными шкалами по нижней и левой вертикальной граням; FRAME — оси с точкой пересечения в левом нижнем углу рисунка; NONE — оси не отображаются
axesfont	Задает шрифт для надписей под засечками вдоль осей координат. Значение этой опции аналогично значению опции font
color	Задает цвета кривых, отображаемых на график. В качестве значения этой опции может выступать одно из зарезервированных значений цвета в Maple: aquamarine, black, blue, navy, coral, cyan, brown, gold, green, gray, grey, khaki, magenta, maroon, orange, pink, plum, red, sienna, tan, turquoise, violet, wheat, white и yellow. Можно также определить и собственный цвет, соответствующий смешению заданных частей красного, зеленого и синего цветов. Это осуществляется с помощью следующей команды macro(palegreen= COLOR(RGB,.5607,.7372,.5607)), где palegreen — имя константы нового цвета, в котором красный составляет 0.5607 части, зеленый 0.7372 и синий 0.5607. В дальнейшем это имя можно использовать для задания цвета аналогично именам встроенных цветов
coords	По умолчанию при выводе как явно заданной функции, так и параметрически заданной функции используется декартовая система координат (cartesian), т. е. задаваемое уравнение кривой рассматривается именно в этой системе координат. Данная опция меняет тип системы координат. Возможные значения: bipolar, cardiod, cassinian, elliptic, hyperbolic, invcassinian, invelliptic, logarithmic, logcosh, maxwell, parabolic, polar, rose и tangent, описание которых можно получить в справочной системе Maple с помощью команды ?coords. Здесь отметим только, что значение polar задает полярную систему координат

Таблица 4.1 (продолжение)

Опция	Описание
discont	Установка значения этой опции, равной true, приводит к тому, что Maple первоначально вызывает команду discont(), которая определяет промежутки непрерывности функции, а затем на них рисуются непрерывные участки графика функции. Значение по умолчанию false
filled	Установка значения данной опции равным true приводит к тому, что область, ограниченная графиком функции и горизонтальной осью *x*, закрашивается заданным в опции color цветом
font	Задает шрифт для вывода текста на рисунке. Значение опции задается в виде списка [семейство, стиль, размер]. Параметр семейство задает гарнитуру шрифта: TIMES, COURIER, HELVETICA или SYMBOL. Параметр стиль определяет стиль шрифта: для гарнитуры TIMES возможные значения ROMAN, BOLD, ITALIC или BOLDITALIC, для гарнитур COURIER и HELVETICA стиль можно опустить или задать BOLD, OBLIQUE или BOLDOBLIQUE, для шрифта SYMBOL стиль не задается. Последний параметр размер задает размер шрифта в пунктах (points) (один пункт приблизительно равен 1/72 дюйма)
labels	Задание названий осей координат в виде списка [x,y]. Параметры x и y задаются в виде строк и соответствуют отображаемым названиям горизонтальной и вертикальной осей. По умолчанию принимают значения имени независимой переменной и имени функции
labeldirections	Эта опция определяет направление отображения названий осей и задается в виде списка [x,y], элементы которого могут принимать одно из двух значений HORISONTAL или VERTICAL и определяют расположение надписей осей координат: горизонтально или вертикально. Умалчиваемое значение HORISONTAL
labelfont	Задает параметры шрифта, которым отображаются названия осей координат. Значение этой опции аналогично значению опции font
legend	Задает отображение легенды для нескольких кривых на одном графике в виде списка, в котором *i*-й строковый элемент соответствует *i*-й кривой графика
linestyle	Определяет тип линии графика. Значением этой опции является целое число n. При n=0 тип линии соответствует умалчиваемому типу для используемого устройства отображения (обычно сплошная линия), значение 1 соответствует сплошной линии, значение 2 — отображению линии точками, 3 — пунктиром и 4 — штрихпунктиром

Таблица 4.1 *(продолжение)*

Опция	Описание
numpoints	Определяет минимальное число вычисляемых точек, по которым строится график (значение по умолчанию равно 50)
resolution	Определяет горизонтальное разрешение дисплея в пикселах на дюйм и используется в качестве критерия для завершения адаптивного алгоритма отображения (значение по умолчанию равно 200)
sample	Определяет список значений параметров, который используется для "пробного" отображения кривой. Отключение адаптивного алгоритма вычисления точек кривой позволяет явным образом управлять отображением кривой
scaling	Задает масштаб, в котором отображается график. Если значение этой опции равно CONSTRAINED, то это соответствует заданию абсолютных значений по осям координат, т. е. одна единица измерения по оси независимой переменной равна одной единице измерения по оси значений функции. Значение по умолчанию равно UNCONSTRAINED, и это соответствует тому, что оси растягиваются таким образом, чтобы их размеры соответствовали размерам графического окна вывода
style	Задает отображение графика функции линиями (значение опции равно LINE) или точками (значение опции равно POINT). Значения этой опции, равные PATCH и PATCHNOGRID, применяются только тогда, когда выводится замкнутый многоугольник (графическая структура POLYGONS). В этом случае его внутренняя область закрашивается цветом, установленным в опции color, причем в случае значения PATCHNOGRID его граница не отображается. Если в графическом выводе нет замкнутых многоугольников, то действие этих значений данной опции соответствует значению LINE
symbol	Определяет тип символа, которым помечаются точки графика функции при style=POINT. Может принимать следующие значения: BOX для □, CROSS для +, CIRCLE для ○, POINT для • (точка) и DIAMOND для ◊
symbolize	Задает размер символа в пунктах. Его значение может быть любое натуральное число. По умолчанию используются символы размером 10 пунктов. Действие этой опции не распространяется на символ POINT
thickness	Задает толщину линии графика. Значение является целым числом и изменяется от 0 до 3, соответствуя изменению толщины линии от самой тонкой до самой жирной

Таблица 4.1 (окончание)

Опция	Описание
tickmarks	Определяет число точек, не менее которого должно быть помечено по горизонтальной и вертикальной оси координат. Значение задается в виде списка [n,m]. Для каждой из осей можно определить список помечаемых точек
title	Определяет строку, которая выводится как заголовок рисунка. По умолчанию заголовок не выводится. В строке можно использовать специальные комбинации символов. Например, \n осуществляет перевод на новую строку, формируя тем самым многострочный заголовок
titlefont	Определяет шрифт для заголовка рисунка. Значение этой опции аналогично значению опции font
xtickmarks	Задает число точек, не менее которого должно быть помечено на горизонтальной оси. Значение этой опции может быть целым числом или списком значений координат точек горизонтальной оси, которые должны быть помечены. Список может состоять из уравнений, левые части которых определяют координаты помечаемых точек, а правые задают в обратных кавычках отображаемый текст, например, [0=`0.`,0.5=`1/2`,1=`1.`]
ytickmarks	Задает число точек, не менее которого должно быть помечено на вертикальной оси. Значение этой опции может быть целым числом или списком значений координат точек вертикальной оси, которые должны быть помечены. Список может состоять из уравнений, левые части которых определяют координаты помечаемых точек, а правые задают в обратных кавычках отображаемый текст, например, [0=`0.`,0.5=`1/2`,1=`1.`]

Работа с командой plot() не представляет никаких сложностей. Несколько примеров позволят легко с ней освоиться.

Первым нашим примером будет отображение графика функции $y(x) = x^2 + \sin(x^2)$ на интервале [-4,4] изменения независимой переменной x с созданием надписи.

Пример 4.1. График функции с надписью

```
> plot(x^2+sin(x^2),x=-4..4,
>                     color=black,
>                     Пример вывода\nграфика",
>                     titlefont=[HELVETICA,12],
>                     xtickmarks=8,
>                     thickness=3,
```

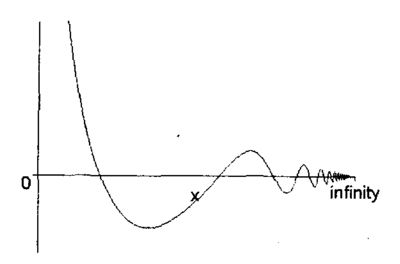
```
>                   axesfont=[COURIER,10],
>                   labels=["x","y(x)],
>                   labeldirections=[HORIZONTAL,VERTICAL],
>                   labelfont=[TIMES,BOLDITALIC,11]);
```



Обратите внимание, что для создания многострочной надписи в строке значения опции `title` использован символ перехода на новую строку (`\n`). Также на графике примера 4.1 мы изменили шрифт надписей вдоль осей, задали их названия и отобразили вертикально название оси `y`.

Команда `plot()` может отображать графики функций не только на конечном интервале изменения независимой переменной, но и на бесконечном:

```
> plot(cos(x)/x,x=0..infinity,-0.5..1,color=black,numpoints=800);
```



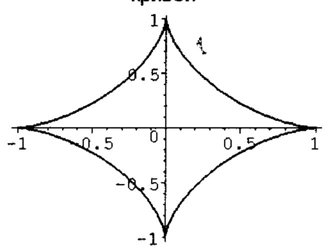
Здесь нам пришлось ограничить область значений функции диапазоном `[-0.5,1]`, так как при `x`, стремящемся к нулю, функция стремится к бесконечности, а также задать больше точек на графике функции, иначе в районе надписи `infinity` не наблюдалась бы гладкость функции, а были бы явные сломы, которые не соответствуют поведению функции.

Не всякую функцию можно представить в явном виде. Многие функции задаются в параметрической форме. Отображение графиков таких функций ничем не отличается от вывода явно задаваемых функций. Единственное отличие заключается в том, что параметрическая кривая задается в виде списка, где первый и второй элементы являются выражениями через параметр, соответственно, горизонтальной и вертикальной координат, а третий

элемент списка задает изменение параметра в виде диапазона Maple. Пример 4.2 демонстрирует отображение параметрически заданной кривой.

Пример 4.2. Отображение графика параметрически заданной функции

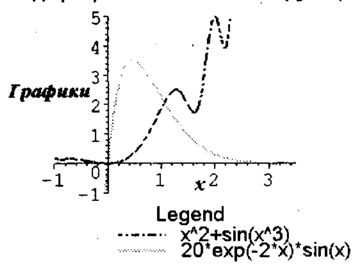
```
> plot([cos(t)^3,sin(t)^3,t=0..2*Pi],
>                   color=[black],
>                   title=`Отображение параметрической\nкривой`,
>                   titlefont=[HELVETICA,11],
>                   xtickmarks=4,
>                   thickness=2,
>                   axesfont=[COURIER,10]);
```



Для вывода нескольких функций на одном графике необходимо в команде `plot()` задавать функции в виде множества или списка, а значение опции `color` в виде списка позволяет задать цвет для вывода графиков функций. Если опция `color` не задана, то Maple отображает функции в соответствии со списком цветов по умолчанию.

Пример 4.3. Отображение графиков нескольких функций

```
> plot([x^2+sin(x^3), 20*exp(-2*x)*sin(x)],
>                   x=-1..3.5, -1..5,
>                   color=[black,green],
>                   title="Вывод графиков нескольких функций",
>                   titlefont=[HELVETICA,11],
>                   legend=["x^2+sin(x^3),"20*exp(-2*x)*sin(x)],
>                   xtickmarks=8,
>                   thickness=2,
>                   linestyle=[4,1],
>                   axesfont=[COURIER,10],
>                   labels=["x","Графики"],
>                   labelfont=[TIMES,BOLDITALIC,10]);
```



При выводе нескольких графиков рекомендуется также отображать легенду заданием списка значений опции `legend`.

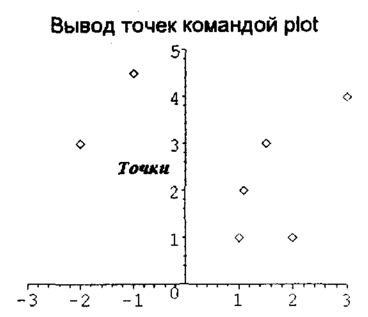
Совет

Легенду всегда можно скрыть или снова отобразить с помощью команды **Show Legend** меню **Legend**.

Команда `plot()` позволяет отображать на графике отдельные точки, которые задаются в виде списка списков, т. е. списка, элементами которого являются списки. Эти двухэлементные списки определяют координаты точек на плоскости. Для вывода точек необходимо задать значение опции `style`, равной `POINT`. Если этого не сделать, то Maple отобразит ломаную линию, соединяющую точки в последовательности их задания, не выделяя их специальными символами. В примере 4.4 точки, заданные своими координатами на плоскости, отображаются с использованием символа ромб `symbol=DIAMOND`.

Пример 4.4. Отображение точек на плоскости

```
> plot([[1,1],[2,1],[3,4],[-2,3],[-1,4.5],[1.5,3],[1.1,2]],
>                     x=-3..3, 0..5,
>                     color=[black],
>                     style=POINT,
>                     symbol=DIAMOND,
>                     symbolsize=18,
>                     title="Вывод точек командой plot",
>                     titlefont=[HELVETICA,11],
>                     xtickmarks=4,
>                     axesfont=[COURIER,10],
>                     labels=["","Точки"],
>                     labelfont=[TIMES,BOLDITALIC,10]);
```



4.1.2. Меню для работы с двумерной графикой

После того как график функции построен командой `plot()` или другой командой двумерной графики из пакета `plots`, можно изменить его внешний вид, переустановив значения некоторых опций с помощью команд основного меню интерфейса пользователя, контекстной панели инструментов или команд контекстного меню, отображаемого нажатием левой кнопки мыши при наведении указателя в область рисунка. Как отмечалось в главе 1, при выделении на рабочем листе графики или получении фокуса окном с графическим выводом (если задан режим вывода графики в отдельном окне) меняется основное меню интерфейса пользователя, а также заменяется контекстная панель инструментов таким образом, чтобы обеспечить доступ к командам интерфейса, работающим с графическим отображением.

На рис. 4.1 показан общий вид интерфейса пользователя с меню и контекстной панелью инструментов для работы с графикой. Также на этом же рисунке отображено контекстное меню, появляющееся при щелчке правой кнопкой мыши, когда указатель расположен в области графического вывода.

При выделении двумерной графики на рабочем листе меню **Insert**, **Spreadsheet** и **Options**, находящиеся в строке основного меню, заменяются новыми: **Style**, **Legend**, **Axes**, **Projection**, **Animation** и **Export**, которые позволяют изменить основные опции построенного графика, а также сохранить его в различных форматах с помощью команд последнего меню. Все команды этих меню дублируются в контекстном меню, в котором дополнительно присутствует команда **Copy** копирования графики в Буфер обмена, а некоторая их часть в контекстной панели инструментов для двумерной графики. На рис. 4.1 показаны опции, которым соответствуют кнопки контекстной панели инструментов. Дополнительно к изменению основных меню сокращается список команд меню **Format**. Так меняется окно интерфейса пользователя при выделении двумерного графика на рабочем листе.

Команды **Line** (Линия), **Point** (Точка), **Patch** (Заливка) и **Patch w/o grid** (Заливка без сетки) меню **Style** устанавливают значение опции `style`, равной, соответственно, `LINE`, `POINT`, `PATCH` или `PATCHNOGRID`. На контекстной панели инструментов этим командам соответствуют первые четыре кнопки (рис. 4.1).

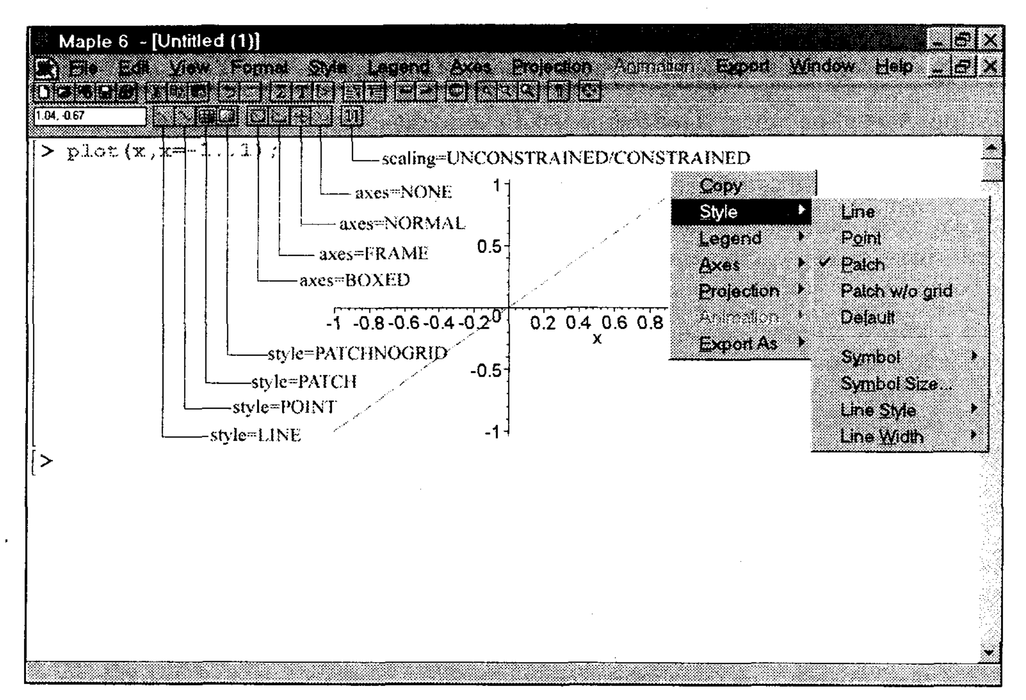


Рис. 4.1. Интерфейс пользователя при выделении графики

Подменю **Symbol** (Символ), **Line Style** (Тип линии) и **Line Width** (Толщина линии) позволяют установить значения опций `symbol`, `linestyle` и `thickness`, а команда **Symbol Size** отвечает за установку и изменение размеров символов отображения точек при отображении линий точками, т. е. когда установлена опция `linestyle=POINT`. Кнопок на контекстной панели инструментов для этих опций не предусмотрено.

Команда **Show Legend** меню **Legend** добавляет или удаляет легенду из выделенного графика. Она работает как переключатель: если легенда помещена на график, то слева от команды отображается "галочка", если на графике легенда отсутствует, то отсутствует и "галочка". Команда **Edit Legend** отображает диалоговое окно **Legend Labels**, в котором можно изменить надписи легенд для кривых, отображаемых на графике. Для этого следует в раскрывающемся списке **Curve** выбрать необходимую кривую, а в поле **Label** ввести новое значение надписи.

Команды меню **Axes** (Оси) позволяют установить значения BOXED, FRAME, NORMAL и NONE опции `axes`. На контекстной панели инструментов им соответствуют следующие четыре кнопки (рис. 4.1).

Меню **Projection** (Проекция) устанавливает значения опции `scaling`. На панели инструментов значения этой опции можно устанавливать с помощью

последней кнопки. Если она не нажата, то соответствует значению UNCONSTRAINED, если нажата — установлено значение CONSTRAINED.

Меню **Animation** специально предназначено для анимации изображений и становится доступным, когда в документе Maple графика выводится командой создания анимации `animate()`.

Командами последнего меню **Export** можно сохранить выделенный на рабочем листе график в одном из следующих форматов: EPS, GIF, JPG, BMP и WMF.

Все перечисленные команды меню можно выполнить из контекстного меню, в котором кроме этих команд присутствует команда **Copy** (Копировать), копирующая графический рисунок в Буфер обмена операционной системы Windows 95/98/NT для вставки его в документ другого приложения или обработки какой-либо графической программой. Отметим, что скопировать график в Буфер обмена можно и командой **Copy** меню **Edit**.

4.1.3. Двумерные команды пакета *plots*

На плоскости кроме прямоугольной декартовой системы координат используются и другие. Одной из наиболее часто применяемой является полярная система координат, в которой положение точки задается также двумя величинами. Они представляют собой длину r радиус-вектора, проведенного из начала координат в заданную точку, и угол наклона φ этого вектора относительно положительного направления горизонтальной оси координат. Многие "замечательные" плоские кривые легче и проще задавать именно в полярной системе координат. Например, окружность радиуса a с центром в начале координат в полярной системе координат задается простым уравнением $r = a$, тогда как в декартовой системе координат эта же окружность задается уравнением в неявном виде $\sqrt{x^2+y^2}=a^2$.

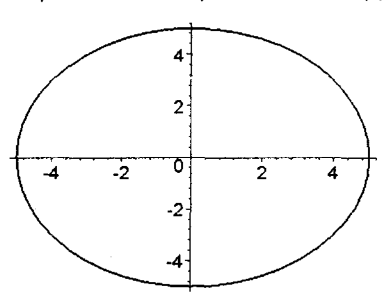
Для отображения графика функции, заданной в полярной системе координат, в пакете `plots` существует функция `polarplot()`. Ее синтаксис похож на синтаксис команды `plot()` за одним исключением — не задается третий параметр, ограничивающий диапазон изменения значений, в данном случае длины радиус-вектора:

```
polarplot(r, phi=диапазон, опции);
```

Параметр `r` — это выражение или функция, зависящие от независимой переменной `phi`, интерпретируемой как угол поворота радиус-вектора относительно горизонтальной оси. Диапазон изменения независимой переменной может отсутствовать, тогда используется диапазон изменения по умолчанию `-Pi..Pi`. Остальные параметры представляют собой такие же опции, что и в функции `plot()`. Использование команды построения окружности в полярной системе координат демонстрируется в примере 4.5.

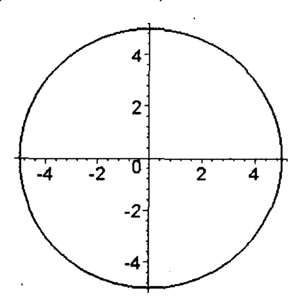
Пример 4.5. График функции в полярной системе координат

```
> polarplot(5,phi=0..2*Pi,color=black,thickness=2);
```



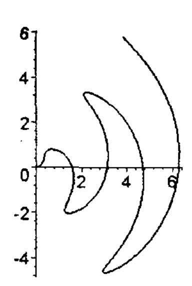
Если посмотреть на вывод этой команды, то вместо обещанной окружности мы видим эллипс. Дело в том, что по умолчанию во всех графических командах используется значение `UNCONSTRAINED` параметра `scaling`. А это означает, что график растягивается по осям таким образом, чтобы полностью заполнить отводимое под него пространство на рабочем листе, что приводит к несоответствию единиц измерения по горизонтальной и вертикальной осям. Подобное явление характерно для вывода *всех* графических команд Maple. Исправить подобный дефект можно с помощью команд интерфейса пользователя или при отображении кривой в соответствующей команде, задав опцию `scaling=CONSTRAINED`:

```
> polarplot(5,phi=0..2*Pi,color=black,thickness=2,scaling=constrained);
```



Команда `polarplot()` также позволяет отображать графики параметрически заданных кривых. Для этого подобную кривую следует задать в форме трехэлементного списка, в котором первые два элемента представляют выражение через параметр длины радиуса-вектора и его угла поворота, а третий элемент задает диапазон изменения параметра:

```
> polarplot([r,sin(2*r),r=0..7],
           color=black,
           thickness=2,
           scaling=constrained);
```



Замечание

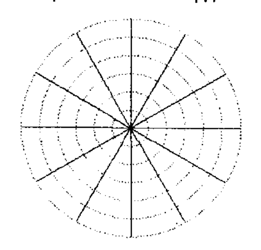
График этой же функции можно построить и командой `plot()`, но в ней следует задать опцию `coords=polar`.

Замечание

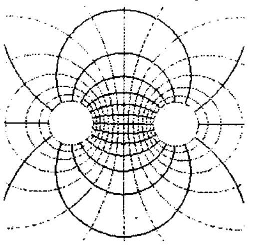
Для отображения командой `polarplot()` на одном графике нескольких кривых, их следует задавать, как и в случае с командой `plot()`, в виде списка.

В Maple командой `coordplot()` можно начертить "линии уровня" плоских систем координат, поддерживаемых командой `plot()` через опцию `coords`. В качестве параметра этой функции передается название системы координат (см. опцию `coords` в табл. 4.1):

```
> coordplot(polar,color=[red,green],scaling=CONSTRAINED,
          title="Полярная система координат");
```




```
> coordplot(bipolar,color=[green,red],scaling=CONSTRAINED,
          title="Биполярная система координат");
```



Замечание

Только для полярной системы координат существует специальная команда построения графика функции. Для отображения графиков функций в других допустимых системах координат следует использовать опцию `coords` команды `plot()`.

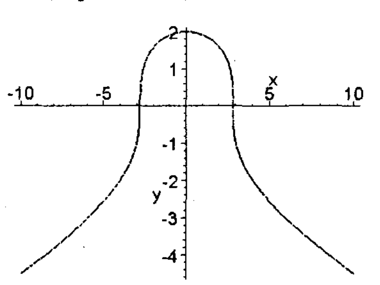
Бывает так, что искомая функция, график которой надо отобразить, представляется только в неявном виде $f(x,y) = 0$ и никакими ухищрениями ее нельзя представить в явной форме ни в одной из известных систем координат. В таком случае следует воспользоваться командой `implicitplot()`, которая специально разработана для отображения неявных функций:

```
implicitplot(expr, x=a..b, y=c..d, опции);
implicitplot(f, a..b, c..d, опции);
```

Здесь в первой форме вызова команды параметр `expr` представляет уравнение, зависящее от двух переменных `x` и `y`, а во второй форме `f` представляет уравнение, и в левую, и в правую части которого входят только процедуры-функции и операторы от двух переменных. Дополнительно ко всем известным опциям команды `plot()` можно задать опцию `grid=[m,n]`, определяющую сетку из `mxn` точек, на которой вычерчивается кривая. При увеличении количества точек в сетке кривая отображается более гладкой без угловых точек. По умолчанию используется сетка `25x25` точек. Опцией `coords` можно задавать график в разных системах координат, по умолчанию используется декартовая прямоугольная система координат.

Пример 4.6. График неявно заданной функции

```
> implicitplot(x^2+y^3-8=0,x=-10..10,y=-8..8,
              color=black, grid=[60,60],thickness=2);
```



Замечание

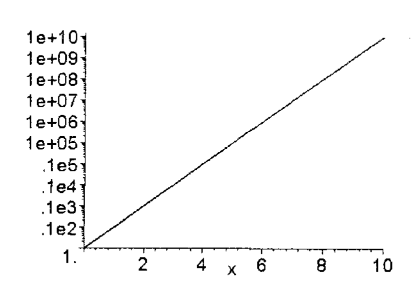
Увеличение числа точек в сетке, на которой рассчитывается неявно заданная кривая, приводит к существенному увеличению времени расчета ее графика.

Часто в технике при отображении зависимости двух параметров какой-либо системы используются логарифмические оси координат, т. е. по одной или

обеим осям откладываются десятичные логарифмы соответствующих величин. В пакете plots содержатся три команды, позволяющие работать с подобным представлением функций: loglogplot() осуществляет логарифмическое преобразование обеих координат, semilogplot() только горизонтальной оси, а logplot() только вертикальной оси координат.

Пример 4.7. Отображение функции 10^x в логарифмической шкале

```
> logplot(10^x,x=0..10,color=black,thickness=2);
```



Очень полезная команда inequal() отображает на плоскости решение смешанной системы линейных неравенств и уравнений двух независимых переменных. Ее синтаксис несколько отличается от синтаксиса большинства команд пакета plots. При обращении к ней для каждой из перечисленных ниже областей, на которые смешанная система делит двумерную плоскость, возможно задание собственных значений опций после указания опции, идентифицирующей соответствующую область:

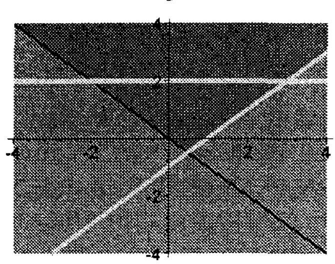
- область, удовлетворяющая системе неравенств (optionsfeasible=(опции));
- область, не удовлетворяющая хотя бы одному неравенству из заданной смешанной системы (optionsexcluded=(опции));
- область границы строгих неравенств (optionsopen=(опции));
- область границы нестрогих неравенств и равенств (optionsclosed=(опции)).

После задания соответствующей опции области в правой части в круглых скобках задаются обычные графические опции, имеющие значение для указанной области, например, цвет области, цвет и толщина линии границы. Пример 4.8 иллюстрирует самое общее использование команды построения решения смешанной системы на плоскости.

Пример 4.8. Отображение решения смешанной системы неравенств и равенств

```
> inequal( { x+y>0, x-y<=1,y=2}, x=-4..4, y=-4..4,
         optionsfeasible=(color=red),
```

```
optionsopen=(color=black,thickness=3),
optionsclosed=(color=green, thickness=6),
optionsexcluded=(color=magenta));
```



Замечание

На рисунке область более темного цвета соответствует области решения неравенств, более светлого — области, не удовлетворяющей системе неравенств. Черная линия представляет область границы строгих неравенств, а толстая светлая линия — область границы нестрогих неравенств и равенств, входящих в смешанную систему.

Команда `polygonplot()` строит на плоскости один или несколько многоугольников, заданных своими вершинами. Каждый многоугольник задается в виде списка координат его вершин, представленных в форме двухэлементных списков. В случае отображения нескольких многоугольников они задаются либо списком, либо множеством.

Пример 4.9. Отображение многоугольников

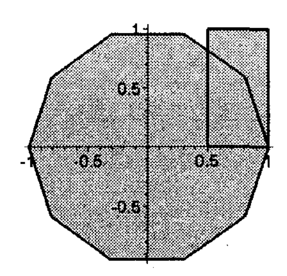
```
> one_poly := [[0.5,0],[0.5,1],[1,1],[1,0]];
```

$$one_poly := [[.5, 0], [.5, 1], [1, 1], [1, 0]]$$

```
> ngon := n -> [seq([ cos(2*Pi*i/n), sin(2*Pi*i/n) ], i = 1..n)];
```

$$ngon := n \to \left[\operatorname{seq}\left(\left[\cos\left(2 \frac{\pi\, i}{n}\right), \sin\left(2 \frac{\pi\, i}{n}\right)\right], i = 1 .. n\right)\right]$$

```
> polygonplot([one_poly,ngon(10)],color=grey,thickness=3,
        scaling=CONSTRAINED);
```

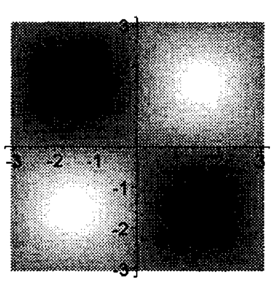


Две двумерные графические команды отображают на плоскости значения функции двух независимых переменных: `densityplot()` — в виде функции

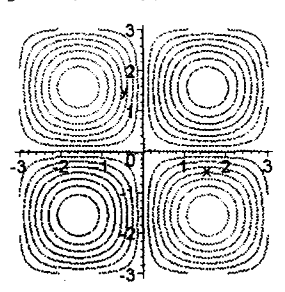
плотности линий уровня (более темные области соответствуют большим значениям функции и наоборот — более светлые представляют меньшие значения функции) и contourplot() — в виде линий уровня. При отображении функции плотности линий уровня по умолчанию отображаются линии сетки, на которой она рассчитывается. Установкой значений опции style, равной patchnogrid, можно отменить отображение линий сетки. Команда contourplot() по умолчанию отображает восемь линий уровня. Изменить это значение можно опцией contours, значением которой является количество линий уровня или список значений линий уровней. Опцией coloring задается цвет линии уровня с наименьшим значением и цвет линии уровня с наибольшим значением. Все остальные отображаются цветом соответствующего оттенка между указанными цветами в двухэлементном списке. Если значение опции filled установлено равным true, то отображается функция плотности линий уровня с указанным цветовым переходом и линии уровня черным цветом.

Пример 4.10. Функция плотности и линии уровня поверхностей

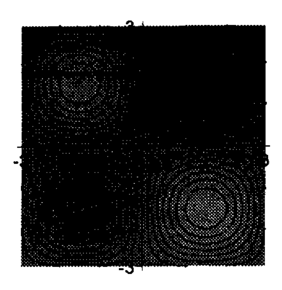
```
> densityplot(sin(x)*sin(y),x=-3..3,y=-3..3,
        grid=[40,40],scaling=CONSTRAINED,style=patchnogrid);
```



```
> contourplot(sin(x)*sin(y),x=-3..3,y=-3..3,
        grid=[40,40],scaling=CONSTRAINED,contours=16,
        coloring=[magenta,blue],thickness=2);
```



```
> contourplot(sin(x)*sin(y),x=-3..3,y=-3..3,
        grid=[40,40],scaling=CONSTRAINED,contours=16,
        coloring=[magenta,blue],filled=true,thickness=2);
```

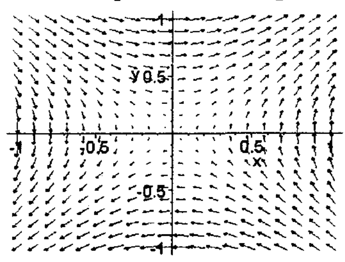


Команды `gradplot()` и `fieldplot()` похожи в том отношении, что обе они отображают векторные поля на плоскости: первая — поле градиентов заданной функции двух переменных, а вторая — простое векторное поле, определяемое координатами векторов в заданных точках поля. Обе эти команды для задания размеров и вида отображаемых векторов используют опцию `arrows`, которая может принимать следующие значения: `THIN` (умалчиваемое значение), `LINE`, `SLIM` или `THICK`. Опцией `color` задается функция двух переменных, которая используется для определения цвета вектора в точке. Векторное поле для команды `fieldplot()` задается в виде двухэлементного списка координат векторов, представленных функциями двух независимых переменных.

Пример 4.11. Отображение поля градиентов функции и векторного поля на плоскости

```
> gradplot(sin(x)*sin(y),x=-3..3,y=-3..3,
        grid=[15,15],arrows=THICK,
        color=sin(x)*sin(y),scaling=CONSTRAINED);
```

```
> fieldplot([y*cos(x*y),x*cos(x*y)],x=-1..1,y=-1..1,arrows=SLIM);
```



Команда `odeplot()` используется для отображения численного решения задачи Коши для дифференциального уравнения или системы дифференциальных уравнений. Ее первым параметром является построенная командой `dsolve()` с опцией `type=numeric` процедура численного решения задачи Коши, а вторым списковым параметром задаются отображаемые по горизонтальной и вертикальной осям величины. Например, чтобы отобразить искомую функцию решения `y(x)`, следует задать список `[x,y(x)]`. Если элементами списка являются неизвестные функции системы дифференциальных уравнений, то будет построен фазовый портрет решения задачи Коши. Можно отобразить на одном графике несколько зависимостей, определяя их в виде списка.

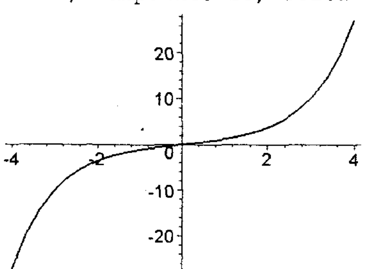
Пример 4.12. Решения дифференциальных уравнений

```
> sys := diff(y(x),x)=z(x),diff(z(x),x)=y(x); # Система диф. уравнений
  fcns := {y(x), z(x)}:                         # Неизвестные функции
  p:= dsolve({sys,y(0)=0,z(0)=1},fcns,type=numeric);
```

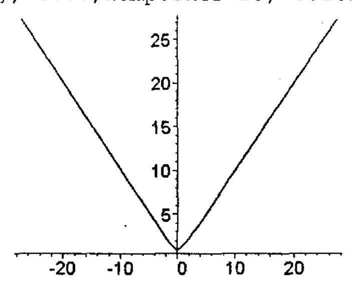
$$sys := \frac{\partial}{\partial x}\,y(x) = z(x),\ \frac{\partial}{\partial x}\,z(x) = y(x)$$

$$p := \textbf{proc}\ (rkf45_x\ )\ ...\ \textbf{end proc}$$

```
> # Зависимость решения y(x) от независимой переменной x.
> odeplot(p, [x,y(x)], -4..4, numpoints=25, color=black,thickness=2);
```




```
> # Фазовый портрет.
> odeplot(p, [y(x),z(x)],-4..4,numpoints=25, color=black,thickness=2);
```

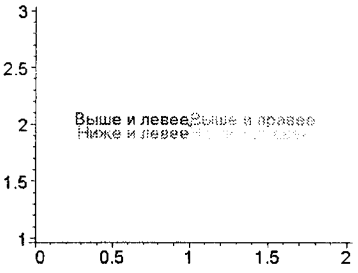


Последняя команда двумерного графического отображения, которую мы опишем, — команда `textplot()` отображения текста в заданной точке гра-

фика. Она отображает так называемые текстовые точки, которые определяются трехэлементным списком. Первые два элемента этого списка являются координатами точки привязки текста на плоскости, а третий элемент — строка выводимого текста. Опцией `align` можно указать расположение текста относительно точки привязки: `ABOVE` — выше, `BELOW` — ниже, `RIGHT` — правее и `LEFT` — левее. Если необходимо задать несколько значений этой опции, то их задают в виде множества. Для отображения нескольких надписей следует задать список текстовых точек в качестве первого параметра функции `textplot()`. В примере 4.13 отображается точка с координатами (1, 2) и привязанный к ней текст с различными значениями опции выравнивания.

Пример 4.13. Отображение текста

```
> t1:=textplot([1,2,"Выше и правее"],align={ABOVE,RIGHT},color=magenta):
> t2:=textplot([1,2,"Выше и левее"],align={ABOVE,LEFT},color=blue):
> t3:=textplot([1,2,"Ниже и левее"],align={BELOW,LEFT},color=red):
> t4:=textplot([1,2,"Ниже и правее"],align={BELOW,RIGHT},color=green):
> f:=plot([[1,2]],style=POINT,color=black,symbolsize=10):
> display([t1,t2,t3,t4,f]);
```



Замечание

Для совмещения на одном рисунке примера 4.13 выводов нескольких графических команд использована команда `display()`, описание которой будет дано ниже в этой же главе.

4.1.4. Двумерные графические структуры Maple

Все двумерные графические команды, как впрочем и трехмерные, преобразуют свою входную информацию в специальные PLOT-структуры данных (в случае трехмерных изображений в PLOT3D-структуры), которые затем преобразуются к формату "установленного" устройства отображения графики и отображаются в нем. В Maple под *устройством отображения графики* понимается тот графический формат, в который следует преобразовать PLOT-

структуру. В зависимости от используемого устройства отображения Maple подключает соответствующий драйвер. Maple 6 поддерживает практически все наиболее популярные графические форматы: GIF, JPEG, DXF, PS, HPGL, HPLJ, WMF, X11, TEK, PCX. При "выводе" на эти устройства формируется графический файл соответствующего формата с именем plot и соответствующим расширением в основной папке системы \Maple 6. Используемые по умолчанию имя и папку расположения формируемого графического файла можно задать в опции `plotoutput` команды `plotsetup()`. Например, после выполнения команды

```
> plotsetup(gif,plotoutput="D:\\gMaple.gif"):
```

вывод всех графических команд будет осуществляться в файл gMaple.gif, расположенный в корневой папке диска D.

Замечание

Для вывода графики в файлы с разными именами следует выполнять команду `plotsetup()` перед каждой графической командой, задавая требуемое имя файла и его расширение.

Для возврата к отображению графики в рабочий лист следует выполнить эту же команду с единственным параметром, принимающим значение `inline` или `default`, для вывода графики в отдельное окно следует использовать значение `window`.

С PLOT-структурой можно осуществлять обычные действия, которые принято выполнять в Maple над разнообразными объектами: ее можно присвоить в качестве значения произвольной переменной, преобразовать в другую структуру, сохранить в файле и даже распечатать на рабочем листе командой `lprint()`:

```
> lprint(plot(x^2,x=0..1,numpoints=5,adaptive=false,filled=true));
PLOT(POLYGONS([[0., 0], [0., 0.], [.261565849999999989,
.684166938862224988e-1], [.261565849999999989, 0]],[[.261565849999999989,
0], [.261565849999999989, .684166938862224988e-1], [.489152897500000016,
.239270557132645528], [.489152897500000016, 0]],[[.489152897500000016,
0], [.489152897500000016, .239270557132645528], [.745098335000000000,
.555171528819772276], [.745098335000000000, 0]],[[.745098335000000000,
0], [.745098335000000000, .555171528819772276], [1., 1.], [1.,
0]],COLOUR(RGB,1.0,0.,0.),STYLE(PATCHNOGRID)),AXESLABELS("x",""),VIEW(0.
.. 1.,DEFAULT))
```

Совет

При присваивании переменной результата выполнения любой графической команды этот оператор следует завершать двоеточием (:), подавляющим вывод результатов выполнения операции, так как в противном случае может выводиться большой объем информации, связанный, в основном, с заданием точек отображаемой кривой или пространственной поверхности.

Как видно из приведенной распечатки PLOT-структуры, сформированной командой `plot()`, она состоит из обращения к функции `PLOT()`. Ее параметрами являются графические структуры, определяющие геометрическую и дополнительную информацию создаваемого графика. Можно "вручную" создать PLOT-структуру, которая немедленно будет прорисована на графическом устройстве отображения. Все, что следует сделать, — это правильно сформировать для функции `PLOT()` графическую информацию.

Для передачи двумерной геометрической информации можно использовать следующие графические структуры:

- `CURVES([[x11,y11],...[x1n,y1n]],[[x21,y21],...[x2k,y2k]],...)` — множество кривых, каждая из которых задается списком, элементами которого являются координаты ее точек. Кривая отображается с помощью линейных сегментов, соединяющих ее точки.
- `POINTS([x1,y1],[x2,y2],...[xn,yn])` — множество точек, заданных своими координатами. Каждая точка отображается с помощью символа, определяемого в структуре `SYMBOL`, соответствующей опции `symbol` графических двумерных команд.
- `POLYGONS([[x11,y11],...[x1n,y1n]],[[x21,y21],...[x2n,y2n]],...)` — множество многоугольников, каждый из которых задается списком, состоящим из координат его вершин.
- `TEXT([x,y],string,horizontal,vertical)` — текст (строковый параметр `string`, который может быть как переменной со строковым значением, так и литеральной строкой), привязанный к точке с координатами `[x,y]`. Параметр `horizontal` может принимать одно из двух значений `ALIGNLEFT` или `ALIGNRIGHT` и соответствует значениям `LEFT` и `RIGHT` опции `align` команды `textplot()`. Параметр может быть либо `ALIGNABOVE`, либо `ALIGNBELOW`, что соответствует значениям `ABOVE` и `BELOW` той же самой команды.

Остальные графические структуры в основном соответствуют используемым в графических командах опциям:

- `AXESLABELS(строка1, строка2)` — надписи при горизонтальной (`строка1`) и вертикальной (`строка2`) осях. Третий необязательный параметр определяет используемый шрифт и задается в виде графической структуры `FONT`.
- `AXESTICKS` — определяет число, расположение и надписи засечек на осях координат. Два ее параметра определяют положение засечек, соответственно, на горизонтальной и вертикальной осях. Параметр может задаваться в виде целого числа, списка чисел, уравнений или специального значения `DEFAULT`, смысл которых полностью соответствует определению засечек в опциях `tickmarks`, `xtickmarks` и `ytickmarks`. Необязательный параметр в виде графической структуры `FONT` определяет используемый для отображения надписей засечек шрифт.

- AXESSTYLE — управляет отображаемым типом осей координат. Единственный параметр может принимать одно из следующих значений: BOX, FRAME, NORMAL, NONE или DEFAULT, которые соответствуют этим же значениям опции axes графических команд.
- COLOR — определяет цвет геометрических объектов и может быть задан тремя способами. Для RGB необходимо задание трех чисел, меньших единицы и определяющих содержание, соответственно, красной, зеленой и синей составляющих цвета: COLOR(RGB, 1.0, 0.0, 0.0) — красный. Для HSV также необходимо задание трех чисел. Дробная часть первого числа определяет цвет, а два оставшихся задают его насыщенность и яркость. Для HUE следует задавать одно число, дробная часть которого определяет цвет: COLOR(HUE, 0.1) — красный.
- FONT — определяет шрифт для отображения геометрических структур TEXT. Требует три параметра для определения семейства шрифта, его гарнитуры и размера в пунктах. Эти параметры полностью соответствуют аналогичным параметрам опции font.
- LINESTYLE — определяет тип линии (сплошная, точечная, пунктирная и штрихпунктирная). Ее единственный параметр представляет целое число и соответствует значениям опции linestyle.
- SCALING — полностью соответствует опции scaling. Допустимые значения единственного параметра CONSTRAINED, UNCONSTRAINED и DEFAULT.
- STYLE — определяет, каким образом отображаются геометрические объекты (кроме TEXT) и полностью идентична опции style. Допустимые значения параметра: POINT, LINE, PATCH, PATCHNOGRID.
- SYMBOL — определяет символ, используемый для отображения точек. Значения его первого параметра соответствуют значениям опции symbol: BOX, CROSS, CIRCLE, POINT, DIAMOND и DEFAULT. Второй необязательный параметр определяет размер символа в пунктах и соответствует опции symbolsize.
- THICKNESS — задает толщину линий при отображении кривых и многоугольников, идентичен опции thickness графических команд.
- TITLE — определяет надпись для графика. Первый параметр задается в виде строки и представляет текст надписи, а второй необязательный параметр задается в виде структуры FONT и определяет шрифт отображения надписи.
- VIEW — задает отображаемую подобласть плоскости графика и имеет два параметра в виде диапазона, задающие, соответственно, интервалы изменения переменных по горизонтальной и вертикальной осям. Можно задать значение DEFAULT, при котором размеры области отображения выбираются так, чтобы все объекты графика были отображены.

Замечание

Все перечисленные графические структуры-опции могут быть использованы в качестве локальных параметров при задании геометрических структур `POINTS`, `CURVES`, `POLYGONS` **и** `TEXT`, **если они имеют для них смысл, и будут перекрывать глобальные установки этих же опций, заданные в функции** `PLOT()`.

Теперь, вооружившись знаниями о параметрах функции `PLOT()`, которая формирует и отображает PLOT-структуру, мы можем непосредственно формировать графики, комбинируя в них разнообразные графические структуры. В примере 4.14 отображается заштрихованный график одной волны синусоиды.

Пример 4.14. Построение графика на базе графических структур

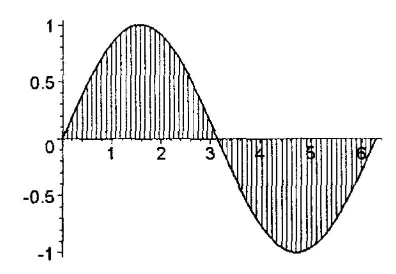
```
> # Формирование двух последовательных точек синусоиды при разбиении
> # интервала [0,2*Pi] на n участков.
> func:=(i,n)->
        [[i*2*Pi/n,sin(i*2*Pi/n)],[(i+1)*2*Pi/n,sin((i+1)*2*Pi/n)]];
```

$$func := (i, n) \to \left[\left[2\,\frac{i\,\pi}{n}, \sin\left(2\,\frac{i\,\pi}{n}\right)\right], \left[2\,\frac{(i+1)\,\pi}{n}, \sin\left(2\,\frac{(i+1)\,\pi}{n}\right)\right]\right]$$

```
> # Формирование нижней и верхней точек штриха на левом конце
> # интервала разбиения
> stroke:=(i,n)->[[i*2*Pi/n,0],[i*2*Pi/n,sin(i*2*Pi/n)]];
```

$$stroke := (i, n) \to \left[\left[2\,\frac{i\,\pi}{n}, 0\right], \left[2\,\frac{i\,\pi}{n}, \sin\left(2\,\frac{i\,\pi}{n}\right)\right]\right]$$

```
> # Задание количества участков разбиения
> n:=64:
> # Отображение синусоиды и штриховки
> PLOT(seq(CURVES(evalf(stroke(i,n))), i=0..n),
      seq(CURVES(evalf(func(i,n)),THICKNESS(2)), i=0..n-1));
```



Для создания плоских геометрических структур можно воспользоваться пакетом `plottools`, который содержит большой набор команд для формирования графических структур наиболее часто используемых двумерных геометрических объектов (табл. 4.2).

Таблица 4.2. *Команды пакета* `plottools` *для двумерных графических структур*

Синтаксис команды	Описание создаваемой графической структуры
`arc([x,y],r,diap,opt)`	Дуга окружности с центром в точке с координатами `x` и `y` и радиусом `r`. Параметр `diap` определяет углы в радианах начальной и конечной точек дуги, отсчитываемые против часовой стрелки. `arc([0,0],2,0..Pi/2)` — первая четверть окружности радиуса 2 с центром в начале координат
`arrow([xb,yb],[xe,ye], wd,wh,hh,opt)`*	Стрелка с началом в точке с координатами (xb, yb) и концом в точке с координатами (xe, ye). Параметр `wd` задает ширину тела стрелки, `wh` ширину головки стрелки и `hh` высоту головки стрелки в частях от ее длины. `Arrow([0,0],[2,0],0.2,0.4,0.3)` — стрелка длиной 2, выходящая из начала координат и направленная в положительную сторону горизонтальной оси, высота головки равна 3/10 ее длины, т. е. 0,6 единицы длины
`curve([[x1,y1],...,[xn,yn]], opt)`*	Кривая, заданная координатами своих точек. Отображается линейными сегментами, соединяющими соседние точки
`disk([x,y],r,opt)`	Круг радиуса `r` с центром в точке с координатами (x,y). Если радиус не задан, то его значение принимается равным 1. По умолчанию круг не закрашивается, опция `color` используется для задания цвета круга. `disk([0,0],color=red)` — красный единичный круг с центром в начале координат
`ellipse([x,y],a,b,opt)`	Эллипс с центром в точке с координатами (x,y) и полуосями `a` и `b`, соответственно. По умолчанию внутренность не закрашивается, установка опции `filled=true` приводит к закрашиванию внутренности цветом, определяемым опцией `color`. Значение опции `numpoints` определяет количество точек на эллипсе. `ellipse([0,0],1,2,filled=true,color=red)` — эллипс с центром в начале координат, полуосями 1 и 2 по горизонтальной и вертикальной осям соответственно и окрашенный в красный цвет

Таблица 4.2 (окончание)

Синтаксис команды	Описание создаваемой графической структуры
`ellipticArc([x,y],a,b,diap, opt)`	Дуга эллипса с центром в точке с координатами `x` и `y` и полуосями `a` и `b`. Параметр `diap` определяет углы в радианах начальной и конечной точек дуги, отсчитываемые против часовой стрелки, и задается в виде диапазона. Если установлена опция `filled=true`, то начальная и конечные точки соединяются и полученная фигура закрашивается цветом, определяемым опцией `color`. `ellipticArc([0,0],1,2,Pi/4..3*Pi/4)` — верхняя часть эллипса
`hyperbola([x,y],a,b,diap, opt)`	Гипербола с центром симметрии в точке (`x`,`y`) и эксцентриситетом $e^2=a^2+b^2$. Гипербола симметрична относительно вертикальной оси. Параметр `diap` определяет диапазон изменения значений гиперболы. `hyperbola([0,0],1,1,-2..2)` — гипербола с центром симметрии, расположенном в начале координат, и эксцентриситетом равным 2
`line([x1,y1],[x2,y2],opt)`*	Отрезок, соединяющий две заданные точки
`pieslice([x,y],r,diap,opt)`	Центральный сектор круга с центром в точке с координатами (`x`,`y`) и радиусом `r`. Параметр `diap` задает диапазон изменения угловой координаты сектора. Опция `color` определяет цвет, которым закрашивается сектор. `pieslice([0,0],1,0..Pi/2)` — первая четверть круга радиуса 1 с центром в начале координат
`point([x,y],opt)`*	Точка с заданными координатами. Опция `color` определяет ее цвет
`polygon([[x1,y1],..., [xn,yn]],opt)`*	Многоугольник, заданный координатами своих вершин. Опция `color` определяет цвет, которым закрашивается внутренность многоугольника
`rectangle([x1,y1],[x2,y2], opt)`	Прямоугольник, задаваемый координатами своих верхней левой и нижней правой точек. Опция `color` определяет цвет, которым закрашивается внутренность прямоугольника

* Создает и трехмерные графические объекты.

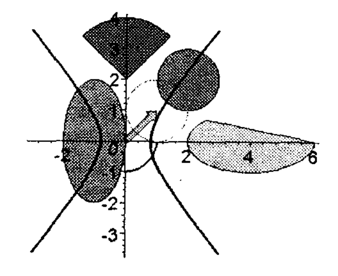
Замечание

Параметр opt во всех командах пакета plottools соответствует допустимым для формируемой структуры опциям графической команды plot(). Для некоторых команд в табл. 4.2 описаны опции, семантика которых несколько отличается от общепринятой.

Пример 4.15 демонстрирует использование команд пакета plottools для формирования и отображения графики.

Пример 4.15. Формирование графических образов командами пакета plottools

```
> a:=arc([0,0], 1, -Pi/2..0,thickness=2):
  arr := arrow([0,0], [1,1], .2, .4, .2, color=grey):
  c := circle([1,1], 1, color=red):
  d := disk([2,2], 1, color=magenta):
  elli := ellipse([-1,0], 1, 2, filled=true, color=gold):
  elliArc := ellipticArc([4,0], 2, 1, 3*Pi/4..2*Pi, filled=true,
                                color=green):
  h := hyperbola([0,0], 0.8, 1, -2..2,thickness=3):
  ps := pieslice([0,2], 2, Pi/4..3*Pi/4, color=red):
  PLOT(a,arr,c,d,elli,elliArc,h,ps,SCALING(CONSTRAINED));
```



Замечание

Отобразить графические структуры можно также командой display() из пакета plots, которая описывается ниже в этом же разделе.

Кроме перечисленных в табл. 4.2 команд создания двумерных графических объектов, пакет plottools содержит ряд команд преобразования произвольных PLOT-структур, в том числе и двумерных. Эти команды выполняют преобразование подобия (homothety()), проекции графического объекта на прямую и плоскость (project()), отображение относительно точки, прямой и плоскости (reflect()), поворота вокруг точки на плоскости и вокруг осей координат и прямой в пространстве (rotate()), масштабирование объекта

относительно заданной точки (`scale()`), параллельный перенос на плоскости и в пространстве (`translate()`) и некоторые другие преобразования. После выполнения соответствующей команды преобразования создается графическая структура, которую можно отобразить на рабочем листе командой `PLOT()` или описываемой ниже командой `display()` из пакета `plots`.

Создавать вручную сложные графики, скомпонованные из разнообразных графических структур, включающих и геометрию кривых, и надписи, достаточно утомительное занятие. Иногда проще получить желаемый результат, совместив на одном графике отображения, генерируемые разными графическими командами. Для этого следует результат выполнения каждой графической команды присвоить некоторой переменной Maple, которая будет содержать уже знакомую нам PLOT-структуру. Заметим, что при этом никакого графического вывода не происходит. Команда `display()`, находящаяся в пакете `plots`, отображает как PLOT-структуры, так и "отложенный" и сохраненный в переменных Maple вывод графических команд. Синтаксис этой команды следующий:

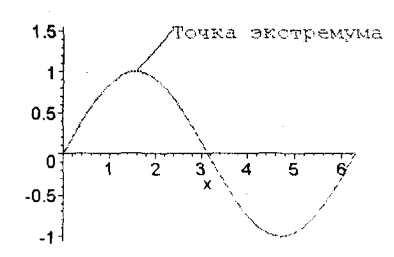
```
display([pic1, pic2, ...], <опции>);
```

Здесь элементы списка являются либо явно заданными графическими образами с помощью команд вывода графики или команд пакета `plottools`, либо переменные, в которых хранятся графические образы. Опции соответствуют рассмотренным ранее опциям команды `plot()`. Следует отметить специальную опцию `insequence`, которая применима только для этой команды. Если ее значение равно `true`, то графические образы `pic1, pic2, ...` будут последовательно отображаться один за другим, составляя в совокупности последовательные кадры анимации. (Анимация рассматривается в разделе 4.3 данной главы.)

Пример 4.16 дает представление о том, как используется команда `display()`. В переменной `graph1` сохраняется график функции `sin(x)`, переменная `graph2` содержит вывод выносной линии для надписи, а переменная `graph3` хранит надпись, выводимую рядом с выносной линией.

Пример 4.16. Построение комбинированного графика

```
> with(plots):
> graph1:=plot(sin(x),x=0..2*Pi,thickness=2):
> graph2:=plot([[Pi/2,1],[Pi/2+Pi/4,1.5]],color=black):
> graph3:=textplot([[Pi/2+Pi/4,1.5,"Точка экстремума"]],
>                    color=blue, align=RIGHT,
>                    font=[COURIER,12]):
> display([graph1,graph2,graph3]);
```



Результирующий график, составленный из этих трех графических изображений, выводится с помощью команды display(), в которой в качестве элементов списка заданы вышеупомянутые переменные, хранящие графические образы.

Замечание

Обратите внимание, что операторы присваивания переменным PLOT-структур завершаются двоеточием, подавляющим, возможно, длинную печать этих структур.

4.1.5. Несколько советов

Завершая разговор о двумерных графических командах пакета plots, мы позволили себе дать несколько советов относительно отображения функций двух классов: разрывных и быстро осциллирующих.

Как известно, функция может иметь в точке разрывы двух видов: первого рода, когда в этой точке происходит скачок в ее значении, и второго рода, когда значение функции при стремлении независимой переменной к заданной точке стремится к плюс или минус бесконечности. Функции, имеющие разрывы только первого рода, обычно называют *кусочно-непрерывными*. Простейшим примером такой функции является ступенчатая функция, которая на интервалах непрерывности принимает постоянные значения, а в нескольких точках своей области задания изменяется скачкообразно на некоторую величину. Примером функции с разрывами второго рода является тригонометрический тангенс.

Maple умеет работать с разрывными функциями, имеющими разрывы любого рода. Для задания кусочно-непрерывных функций следует использовать команду piecewise() со следующим синтаксисом:

```
> piecewise(условие1,значение1, условие2,значение, ...,
            условиеn,значениеn, значение-иначе);
```

Параметры этой команды идут парами и определяют интервал изменения независимой переменной в виде булева выражения условиеn и значение функции на этом интервале значениеn, которое является выражением от не-

зависимой переменной. Последний параметр значение-иначе определяет вид функции на оставшихся интервалах вещественной оси. Например, чтобы задать функцию

$$f(x) = \begin{cases} -1 & x \le 1 \\ x^2 & -x < -1 \text{ and } x < 2 \\ 3 & \textit{otherwise} \end{cases}$$

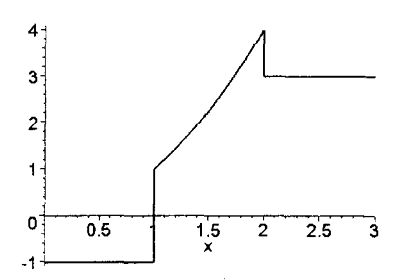
следует выполнить команду:

```
> f:=x->piecewise(x<=1,-1, 1<x and x<2,x^2, 3);
```

$$f := x \to \text{piecewise}(x \le 1, -1, 1 < x \text{ and } x < 2, x^2, 3)$$

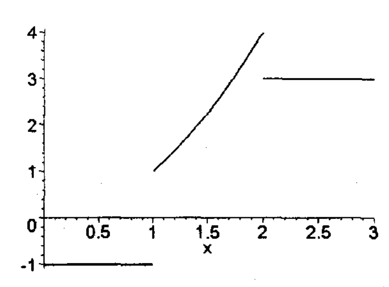
Отобразим график этой функции командой

```
> plot(f(x),x=0..3,color=black,thickness=2);
```



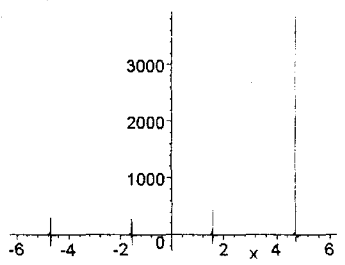
Maple вычерчивает вертикальные линии в точках разрыва, соединяя значения функции справа и слева от точки разрыва. Чтобы избежать такого некорректного отображения разрывной функции, следует при ее вычерчивании устанавливать значение true опции discont:

```
> plot(f(x),x=0..3,color=black,thickness=2,discont=true);
```



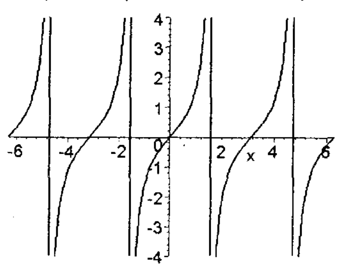
Графики функций с разрывами второго рода следует строить с *обязательным* заданием интервала изменения ее значений. Попытка построить график такой функции командой plot() без ограничения на значения функции обречена на провал:

```
> plot(tan(x),x=-2*Pi..2*Pi, color=black);
```



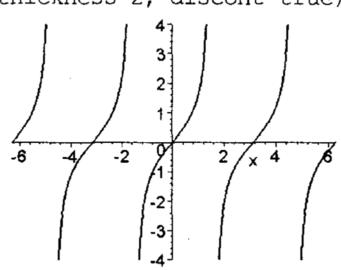
Тогда как ограничения на область изменения значений функции дадут нам правильный график функции с разрывами второго рода:

```
> plot(tan(x),x=-2*Pi..2*Pi,-4..4, color=black,thickness=2);
```



И здесь опять для удаления из графика функции с разрывами второго рода вертикальных прямых в точках разрыва следует, как и в случае с кусочно-непрерывными функциями, воспользоваться опцией `discont`:

```
> plot(tan(x),x=-2*Pi..2*Pi,-4..4, color=black,
>                               thickness=2, discont=true);
```



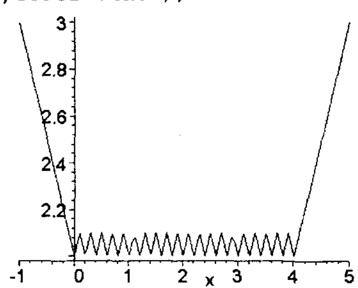
Maple вычерчивает графики функций, вычисляя их значения на некотором множестве эквидистантных точек независимой переменной. Затем он анализирует изменение функции на полученных интервалах и принимает решение вычислить значения функции в дополнительных точках тех интервалов, где функция является быстро осциллирующей. Однако не во всех случаях этот адаптивный алгоритм срабатывает, и приходится прибегать к явному заданию количества точек на графике, чтобы получить его приемлемое отображение.

Отобразим график быстро осциллирующей функции с использованием только адаптивного алгоритма Maple:

```
> s:=Sum((-1)^i*abs(x-i/10),i=0..40);
```

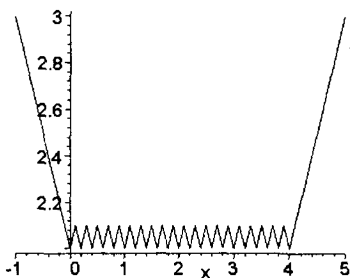
$$s := \sum_{i=0}^{40} (-1)^i \left| -x + \frac{1}{10} i \right|$$

```
> plot(value(s),x=-1..5,color=black);
```



Как видно, зубчики на графике не одинаковой величины, что является свидетельством недостаточности вычисленных адаптивным алгоритмом точек на графике. Улучшить график этой функции поможет задание опции `numpoints`:

```
> plot(value(s),x=-1..5,color=black, numpoints=600);
```



4.2. Пространственная графика

4.2.1. Команда *plot3d()*

Функцию двух переменных можно отобразить как поверхность в трехмерном пространстве, две оси которого соответствуют значениям двух независимых переменных, а по третьей оси откладываются значения функции. В Maple подобную процедуру визуализации функции двух переменных выполняет команда `plot3d()`, которая, как и команда отображения графика функции одной переменной `plot()`, расположена в стандартной библиотеке, а поэтому доступна пользователю в любой момент. Эта команда позволяет

отображать графики функций, заданных как в явном виде, так и в параметрическом виде.

Синтаксис команды plot3d() практически полностью соответствует синтаксису команды plot() с небольшим очевидным дополнением, связанным с наличием второй независимой переменной:

```
plot3d(expr,x=a..b,y=c..d,опции)
```

Здесь параметр expr представляет алгебраическое выражение или обращение к пользовательской функции двух переменных с диапазонами изменения, определяемыми вторым и третьим параметром, в которых вместо x и y следует задавать имена переменных. Пользовательскую функцию можно определять непосредственно в команде, но в этом случае задавать имена переменных не надо:

```
> plot3d((x,t)->cos(x)*sin(t),-1..1,-1..1);
```

Отметим, что и выражение, и функция, представляющие параметр expr, не должны содержать неопределенных символьных переменных, кроме двух упомянутых независимых переменных. Границы диапазонов представляются числами, хотя для второй независимой переменной они могут быть выражениями, зависящими от первой переменной. В этом случае график функции двух переменных отображается не на прямоугольной области, а на четырехугольной, у которой две противоположные границы являются криволинейными. Например, следующая команда

```
> plot3d(cos(x)*sin(t),x=-1..1,t=-5..x^2);
```

отображает график функции на области, у которой одна из границ представлена параболой.

Опции для команд трехмерной графики определяются так же, как и для команд графического отображения на плоскости, в виде уравнения, в левой части которого стоит имя опции, а в правой ее значение. Многие опции команд пространственной графики полностью соответствуют своим двумерным аналогам, правда в некоторых опциях добавлена дополнительная функциональность (смотри, например, опцию color), но есть и специальные опции, отражающие специфику пространственной графики. В табл. 4.3 перечислены все опции с их кратким описанием и возможными значениями, причем значения по умолчанию выделены подчеркиванием.

Таблица 4.3. *Опции команд трехмерной графики*

Опция	Описание
ambientlight	Задает цвет внешнего источника интенсивностями его красной, зеленой и синей составляющих, представленными трехэлементным списком вещественных чисел из интервала [0,1]. Эта опция имеет значение при определении пользователем собственной схемы освещенности поверхности

Таблица 4.3 *(продолжение)*

Опция	Описание
axes	Определяет тип отображаемых осей координат. Эта опция может принимать следующие значения: NORMAL — обычные оси координат, пересекающиеся в точке (0,0,0); BOXED — поверхность заключена в охватывающий параллелепипед с нанесенными шкалами по трем граням; FRAME — три оси отображаются по внешним граням охватывающего параллелепипеда; NONE — оси не отображаются
axesfont	Задает шрифт для надписей под засечками вдоль осей координат. Значение этой опции аналогично значению опции font
color	Задает цвет отображаемой поверхности в случае ее закрашивания или цвет линий сетки в случае отображения поверхности в виде каркаса или линий уровня. В качестве значения этой опции может выступать одно из зарезервированных значений цвета в Maple: aquamarine, black, blue, navy, coral, cyan, brown, gold, green, gray, grey, khaki, magenta, maroon, orange, pink, plum, red, sienna, tan, turquoise, violet, wheat, white и yellow. Можно также определить и собственный цвет, соответствующий смешению заданных частей красного, зеленого и синего цветов. Это осуществляется с помощью следующей команды macro(palegreen=COLOR(RGB,.5607,.7372,.5607)), где palegreen имя константы нового цвета, в котором красный составляет 0.5607 части, зеленый 0.7372 и синий 0.5607. Задание значения в форме выражения, содержащего две независимые переменные, определяет цвет в каждой точке поверхности по цветовой схеме HUE. Если цвет определяется с помощью процедуры, то она должна иметь два аргумента и возвращать значение цвета
contours	Определяет количество линий уровня при отображении их на поверхности или список значений отображаемых линий уровня. Значение по умолчанию равно 10
coords	По умолчанию при выводе как явно заданной функции, так и параметрически заданной функции используется декартовая система координат (cartesian), т. е. задаваемое уравнение кривой рассматривается именно в этой системе координат. Данная опция меняет тип системы координат. Возможные значения: bipolarcylindrical, bispherical, cardioidal, cardioidcylindrical, casscylindrical, confocalellip, confocalparab, conical, cylindrical, ellcylindrical, ellipsoidal, hypercylindrical, invcasscylindrical, invellcylindrical, invoblspheroidal, invprospheroidal,

Таблица 4.3 *(продолжение)*

Опция	Описание
(прод.)	logcoshcylindrical, logcylindrical, maxwellcylindrical, oblatespheroidal, paraboloidal, paracylindrical, prolatespheroidal, rosecylindrical, sixsphere, spherical, tangentcylindrical, tangentsphere и toroidal, описание которых можно получить в справочной системе Maple с помощью команды ?coords
filled	Установка значения данной опции равным true приводит к тому, что область, ограниченная поверхностью и плоскостью *xy*, отображается как твердое непрозрачное тело и закрашивается в соответствии с используемой цветовой схемой
font	Задает шрифт для вывода текста на рисунке. Значение опции задается в виде списка [семейство, стиль, размер]. Параметр семейство задает гарнитуру шрифта: TIMES, COURIER, HELVETICA или SYMBOL. Параметр стиль определяет стиль шрифта: для гарнитуры TIMES возможные значения ROMAN, BOLD, ITALIC или BOLDITALIC, для гарнитур COURIER и HELVETICA стиль можно опустить, или задать BOLD, OBLIQUE или BOLDOBLIQUE, для шрифта SYMBOL стиль не задается. Последний параметр задает размер шрифта в пунктах (points) (один пункт приблизительно равен 1/72 дюйма)
grid	Эта опция определяет прямоугольную равномерную сетку значений независимых переменных отображаемой функции, на которой вычисляются ее значения для построения поверхности и задается в виде двухэлементного списка [m,n], в котором каждый элемент является целым числом, определяющим количество точек по соответствующей координате. По умолчанию используется сетка [25,25]
gridstyle	Задает тип отображаемой сетки: составленной из прямоугольников или треугольников на основании вычисленных ее значений в соответствии с установками, определяемыми опцией grid. Она может принимать одно из двух значений: RECTANGULAR и TRIANGULAR
labels	Задание названий осей координат в виде списка [x,y,z]. Параметры x, y и z задаются в виде строк и соответствуют отображаемым названиям трех осей декартовой системы координат. По умолчанию оси не подписываются
labeldirections	Эта опция определяет направление отображения названия осей и задается в виде списка [x,y,z], элементы которого могут принимать одно из двух значений HORISONTAL или VERTICAL
labelfont	Задает шрифт, которым отображаются названия осей координат. Значение этой опции аналогично значению опции font

Таблица 4.3 *(продолжение)*

Опция	Описание
light	Эта опция определяет расположение и цвет направленного источника света при задании пользовательской схемы подсветки. Ее значения задаются в виде списка [phi, theta,r,g,b], элементы которого имеют следующий смысл: phi и theta определяют углы направления, из которого исходит направленный свет (задаются в сферической системе координат), а r, g, b задают числовые интенсивности красной, зеленой и синей составляющих цвета источника
lightmodel	Эта опция позволяет выбрать одну из предопределенных схем подсветки и может принимать следующие значения: 'none' (нет подсветки), 'light1', 'light2', 'light3' или 'light4'
linestyle	Определяет тип линии на поверхности. Значением этой опции является целое число n. При n=0 или 1 линии отображаются как сплошные, значение, равное 2, соответствует отображению линий точками, 3 — пунктиром и 4 — штрихпунктиром
numpoints	Определяет минимальное количество n вычисляемых точек, по которым строится поверхность (значение по умолчанию равно 625=25^2). Эта опция перекрывает задание сетки опцией grid, определяя ее новые значения по осям независимых переменных, равными равномерно распределенным sqrt(n) точкам в диапазонах изменения независимых переменных
orientation	Задает углы в сферической системе координат направления, из которого пользователь смотрит на отображаемую поверхность: [theta,phi]. Углы задаются в градусах и по умолчанию равны [45°,45°]
projection	Задает проекцию, в которой отображается поверхность. Значением этой опции может быть целое число r из диапазона [0,1], причем 0 соответствует широкоугольной перспективе, 1 — ортогональной проекции, промежуточные значения разным типам перспектив. Можно использовать три зарезервированных ключевых слова: 'FISHEYE' для широкоугольной перспективы (r=0), 'NORMAL' для перспективной проекции с параметром r=0.5 и 'ORTHOGONAL' для ортогональной проекции (r=1)
scaling	Задает масштаб, в котором отображается поверхность. Если значение данной опции равно CONSTRAINED, то это соответствует заданию абсолютных значений по осям координат, т. е. одна единица измерения по оси независимой переменной равна одной единице измерения по оси значений функции. Значение по умолчанию равно UNCONSTRAINED, и это соответствует тому, что оси растягиваются таким образом, чтобы их размеры соответствовали размерам графического окна вывода

Таблица 4.3 *(продолжение)*

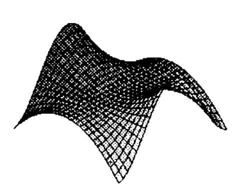
Опция	Описание
shading	Определяет, какая схема закрашивания применяется при отображении поверхности. Допустимые значения: XYZ (цвет точки поверхности зависит от значений трех ее координат), XY (цвет точки поверхности зависит от значений ее двух независимых координат), Z (цвет точки поверхности зависит от значения функции: минимальное представляется синим цветом, максимальное — красным, остальные оттенками при переходе от синего к красному), ZGRAYSCALE (цвет точки поверхности зависит от значения функции: минимальное представляется черным цветом, максимальное — бледно-серым, остальные оттенками при переходе от черного к бледно-серому), ZHUE (цвет точки поверхности зависит от значения функции: от минимального сиреневого через синий, зеленый и желтый к максимальному красному) и NONE (поверхность не закрашена)
style	Определяет, как будет отображаться поверхность. Допустимые значения: POINT (точками, представляющими значения функции на сетке), HIDDEN (каркасная модель с удалением невидимых линий), PATCH (закрашенная поверхность с линиями сетки), WIREFRAME или LINE (каркасная модель без удаления невидимых линий), CONTOUR (линиями уровня), PATCHNOGRID (закрашенная поверхность без линий сетки), PATCHCONTOUR (закрашенная поверхность с линиями уровня)
symbol	Определяет тип символа, которым помечаются точки поверхности функции при опции style=POINT. Может принимать следующие значения: BOX для □, CROSS для +, CIRCLE для ○, POINT для • (точка) и DIAMOND для ◊
symbolsize	Задает размер символа в пунктах. Его значением может быть любое натуральное число. По умолчанию используются символы размером 10 пунктов. Действие этой опции не распространяется на символ точка, задаваемый опцией symbol=POINT
thickness	Задает толщину линий на поверхности. Значение является целым числом и изменяется от 0 до 3, соответствуя изменению толщины линии от самой тонкой до самой жирной
tickmarks	Определяет число точек, не менее которого должно быть помечено по горизонтальной и вертикальной оси координат. Значение задается в виде списка [l,n,m]
title	Определяет строку, которая выводится как заголовок рисунка. По умолчанию заголовок не выводится. В строке можно использовать специальные комбинации символов. Например, \n осуществляет перевод на новую строку, формируя тем самым многострочный заголовок

Таблица 4.3 (окончание)

Опция	Описание
titlefont	Определяет шрифт для заголовка рисунка. Значение этой опции аналогично значению опции font
view	Определяет минимальное и максимальное значения координаты *z*, между которыми отображается поверхность, а также диапазоны изменения независимых координат в виде [xmin..xmax, ymin..ymax, zmin..zmax]. По умолчанию отображается вся поверхность без обрезания в пределах заданных диапазонов изменения независимых переменных в команде plot3d()

Как видно из описания команды plot3d() и ее опций, работа с ней интуитивно ясна и проста. Например, для отображения графика функции $z = \cos(x)y^2$ с заголовком и значениями опций по умолчанию следует выполнить команду:

```
> plot3d(f(x,y),x=-3..3,y=-3..3,
            title="График функции\nz=cos(x)*y^2");
```



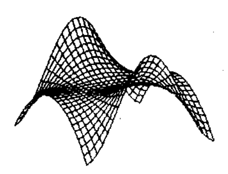
По умолчанию поверхность закрашивается в соответствии с цветовой схемой XYZ, которая выбирает цвет точки поверхности в зависимости от значений трех ее координат.

Замечание

В книге все рисунки, к сожалению, напечатаны с использованием оттенков серого цвета, и читатель не сможет оценить преимущества цветовых гамм Maple. Мы рекомендуем самостоятельно проделать все примеры, чтобы увидеть рисунки в цвете.

Можно отобразить ту же поверхность в виде каркасной модели с удаленными невидимыми линиями и явно заданным направлением взгляда на нее:

```
> plot3d(f(x,y),x=-3..3,y=-3..3,style=hidden,color=black,
     orientation=[60,65],title="График функции\nz=cos(x)*y^2");
```



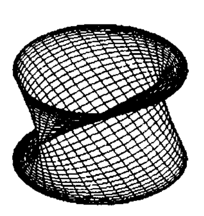
Командой `plot3d()` можно отображать параметрически заданные поверхности. Только надо помнить, что для параметризации трехмерной поверхности следует использовать два параметра, т. е. задать три координаты точек поверхности как функции или выражения двух переменных. Синтаксис команды `plot3d()` в этом случае будет иметь следующий вид:

```
plot3d([x-expr,y-expr,z-expr],диапазон-парам1,диапазон-парам2,опции)
```

Пример 4.17 иллюстрирует отображение параметрически заданной поверхности в декартовой системе координат.

Пример 4.17. Отображение параметрически заданной поверхности

```
> plot3d([sin(x),cos(x)*sin(y),sin(y)],
        x=-Pi..Pi,y=-Pi..Pi,style=hidden,color=black,grid=[40,40]);
```



Для улучшения вида отображаемой поверхности мы явно задали сетку точек, на которой вычисляются значения параметрически заданной функции поверхности.

4.2.2. Меню для работы с трехмерной графикой

Как и в случае с двумерной графикой, после построения командой `plot3d()` или другой командой пространственной графики из пакета `plots` пространственного образа функции двух переменных можно изменить его внешний вид, переустановив значения некоторых опций с помощью команд основного меню интерфейса пользователя, контекстной панели инструментов или команд контекстного меню, предварительно выделив график на рабочем листе. Напомним, что при этом меняется основное меню интерфейса пользователя, а также заменяется контекстная панель инструментов таким обра-

зом, чтобы обеспечить доступ к командам интерфейса, работающим с пространственным графическим отображением.

На рис. 4.2 показан общий вид интерфейса пользователя с меню пространственной графики, контекстной панелью инструментов и контекстным меню при выделении на рабочем листе трехмерного изображения.

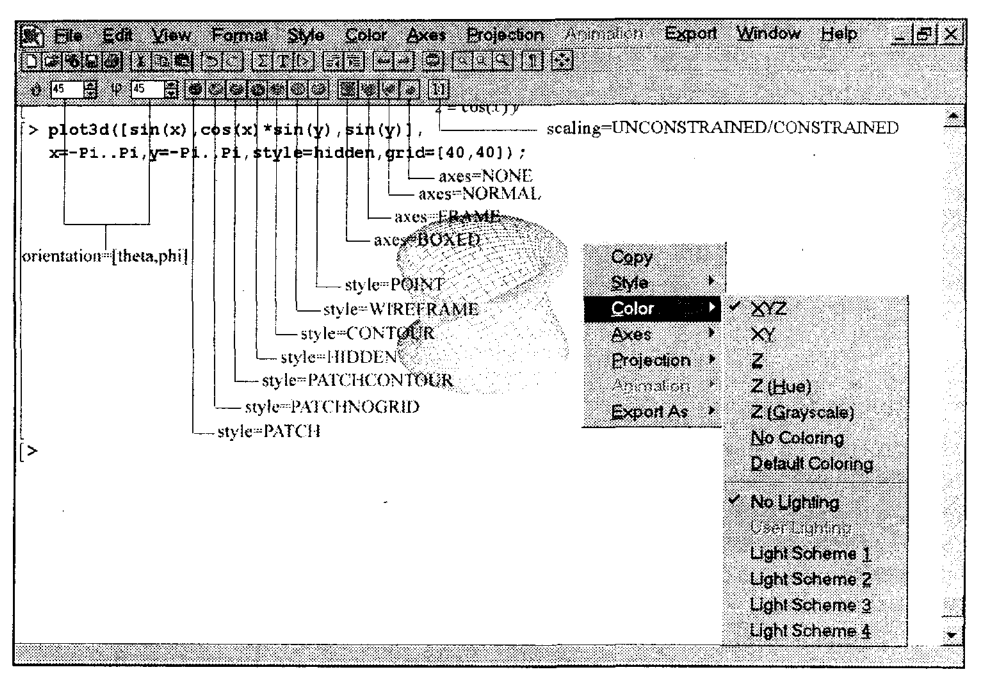


Рис. 4.2. Интерфейс пользователя при выделении трехмерного изображения

При выделении пространственной графики на рабочем листе меню **Insert**, **Spreadsheet** и **Options** строки основного меню, заменяются новыми: **Style**, **Color**, **Axes**, **Projection**, **Animation** и **Export**, которые позволяют изменить основные опции построенного графика, а также сохранить его в различных форматах с помощью команд последнего меню. Все команды этих меню дублируются в контекстном меню, в котором дополнительно присутствует команда **Copy** для копирования графики в Буфер обмена. Некоторые команды можно вызвать нажатием соответствующих кнопок контекстной панели инструментов для работы с пространственной графикой. На рис. 4.2 показаны опции, которым соответствуют кнопки контекстной панели инструментов. Дополнительно к изменению основных меню, как и в случае с двумерной графикой, сокращается список команд меню **Format**. Далее кратко остановимся на всех командах меню трехмерной графики.

Команды меню **Style** разделены на две большие группы. В первой группе (верхняя часть меню) сосредоточены команды, влияющие на способ ото-

бражения поверхности. Они работают как группа переключателей — только одна из них может быть выбрана. При этом слева от нее отображается "галочка". Эти команды меняют значение опции style на указанное в скобках после имени команды: **Patch** (PATCH), **Patch w/o grid** (PATCHNOGRID), **Patch and contour** (PATCHCONTOUR), **Hidden line** (HIDDEN), **Contour** (CONTOUR), **Wireframe** (WIREFRAME), **Point** (POINT) и **Default** (значение по умолчанию, обычно PATCH). На контекстной панели инструментов этим командам соответствуют первые семь кнопок (рис. 4.2).

Команды подменю **Symbol** (Символ), **Line Style** (Тип линии) и **Line Width** (Толщина линии) устанавливают значения опций symbol, linestyle и thickness, а команда **Symbol Size** отвечает за установку и изменение размеров символов отображения точек при изображении линий поверхности точками. Они полностью соответствуют аналогичным подменю из меню работы с плоской графикой. При работе с пространственной графикой в меню **Style** добавляется еще одно подменю **Grid Style**, которое позволяет выбрать отображаемую на поверхности сетку: прямоугольную (команда **Rectangular**) или треугольную (команда **Triangular**), что соответствует установке значений опции gridstyle. Кнопок на контекстной панели инструментов для этих опций не предусмотрено.

Команды меню **Color** меняют цветовую схему закрашивания отображаемой поверхности и схему подсветки поверхности. Группа команд установки цветовой схемы закрашивания изменяет значение опции shading (в скобках указано значение этой опции): **XYZ** (XYZ), **XY** (XY), **Z** (Z), **Z(Hue)** (ZHUE), **Z(Grayscale)** (ZGRAYSCALE), **No Coloring** (NONE) и **Default Coloring** (умалчиваемая схема, зависит от устройства отображения, для рабочего листа XYZ). Вторая группа команд этого меню позволяет выбрать либо предопределенную схему подсветки: **Light Scheme 1** (light1), **Light Scheme 2** (light2), **Light Scheme 3** (light3), **Light Scheme 4** (light4), либо установленную пользователем: **No Lighting** (none), либо вообще отключить подсветку поверхности: **User Lighting**. В скобках указаны значения опции lightmodel, на установку которой влияют перечисленные команды подсветки. Команда **User Lighting** становится доступной только тогда, когда при создании трехмерного отображения командой plot3d() пользователь явным образом указал значения опций ambientlight и light, которые в совокупности определяют схему подсветки.

Команды меню **Axes** позволяют установить значения BOXED, FRAME, NORMAL и NONE опции axes. На контекстной панели инструментов им соответствуют четыре кнопки из группы кнопок, следующей за кнопками управления типом представления поверхности (рис. 4.2).

В меню **Projection** также выделены две группы команд: одна устанавливает значения опции projection, а вторая предназначена для опции scaling. Первая группа команд устанавливает разные типы перспективной проекции для отображения трехмерных поверхностей (в скобках указано значение r

опции `projection`): **No Perspective** (1), **Near Perspective** (0.2), **Medium Perspective** (0.5) и **Far Perspective** (0.8). Для опции `scaling` предусмотрено две команды, устанавливающие ее два значения: **Constrained** (CONSTRAINED) и **Unconstrained** (UNCONSTRAINED). На панели инструментов значения этой опции можно устанавливать с помощью последней кнопки. Если она не нажата, то соответствует значению UNCONSTRAINED, если нажата — установлено значение CONSTRAINED.

Меню **Animation** специально предназначено для анимации изображений и становится доступным, когда в документе Maple пространственная графика выводится командой создания анимации `animate3d()` или `display()`.

Командами последнего меню **Export** можно сохранить выделенный на рабочем листе график в одном из следующих форматов: DXF, EPS, GIF, JPG, POV, BMP и WMF.

Интерфейс пользователя для работы с пространственной графикой позволяет быстро изменить направление взгляда на отображаемую поверхность, т. е. посмотреть на поверхность с разных точек зрения. Для этого достаточно разместить указатель мыши в области пространственного рисунка и, удерживая нажатой левую кнопку мыши, перемещать его. Поверхность будет вращаться и пользователь может выбрать наиболее удачный ракурс ее отображения. При этом в двух левых полях контекстной панели инструментов будут отображаться углы направления, из которого пользователь смотрит на поверхность. Эти два угла определяются в сферической системе координат и имеют значения в градусах. Можно точно установить направление взгляда, воспользовавшись счетчиками, расположенными справа от полей отображения углов, или установить их значения непосредственно в полях.

4.2.3. Трехмерные команды пакета *plots*

В пространстве кроме декартовой системы координат используются и другие (см. значения опции `cords` в табл. 4.3). Наиболее часто применяются цилиндрическая и сферическая системы координат. В пакете `plots` предусмотрены специальные команды, отображающие графики функций двух независимых переменных в этих системах координат: `cylinderplot()` и `sphereplot()`.

В цилиндрической системе координат положение точки задается углом поворота θ проекции ее радиус-вектора на плоскость xy относительно положительного направления оси x, длиной r этой проекции и значением координаты z точки. Команда `cylinderplot()` отображает поверхность, заданную либо в виде явной функции, выражающей зависимость координаты r от двух других θ и z, либо в параметрическом виде, при котором каждая из координат определяется как функция двух параметров. В случае явного задания функции команда имеет следующий синтаксис:

```
cylinderplot(r-exp, theta=диапазон, z=диапазон)
```

Здесь первый аргумент r-exp является выражением от двух переменных theta и z и представляет явный вид задания функции. Для параметрической функции используется другая ее форма, в которой первый аргумент является трехэлементным списком, представляющим зависимость трех координат поверхности в цилиндрической системе координат через два параметра, а следующие два аргумента определяют диапазон изменения параметров поверхности:

```
cylinderplot([r-exp,theta-expr,z-expr], param1=диапазон, param2=диапазон)
```

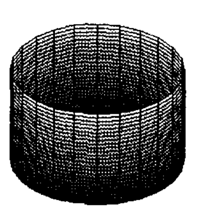
Как и во всех графических командах, кроме указанных аргументов можно использовать любые опции трехмерной графики. Пример 4.18 демонстрирует построение поверхности в цилиндрической системе координат.

Замечание

Следует не забывать подключать пакет plots при обращении ко всем командам данного раздела. В наших примерах мы предполагаем, что он подключен.

Пример 4.18. Построение поверхности в цилиндрической системе координат

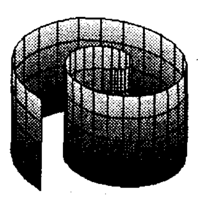
```
> # Круговой цилиндр радиуса 1 и высотой 2.
> cylinderplot(1,theta=0..2*Pi,z=-1..1,shading=ZGRAYSCALE);
```



```
> # Параметрически заданная поверхность
> cylinderplot([s*t,t,cos(s^2)],t=0..Pi,s=-2..2, shading=ZGRAYSCALE);
```



```
> # Спиральный цилиндр высотой 2
> cylinderplot(t,t=0..4*Pi,s=-1..1, shading=ZGRAYSCALE,grid=[50,5]);
```



Обращаем внимание читателя, что для гладкого отображения спирального цилиндра пришлось установить сетку с пятьюдесятью точками по угловой координате θ и пятью точками по линейной координате z.

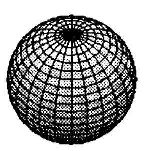
В сферической системе координат положение точки определяется двумя углами и одним линейным размером. Первый угол θ, как и в цилиндрической системе координат, задает угол поворота проекции радиус-вектора точки на плоскость xy. Второй угол — это угол ϕ, который образует радиус-вектор точки с положительным направлением оси z декартовой системы координат. Линейная координата r представляет длину радиус-вектора точки. При работе с командой `sphereplot()`, как и в случае с командой построения поверхностей, заданных в цилиндрической системе координат, возможно либо явное задание поверхности, либо параметрическое. В первом случае необходимо в качестве первого аргумента передать выражение длины радиус-вектора через угловые координаты и задать их диапазоны изменения, во втором случае следует задать список сферических координат точек поверхности в форме выражений от двух параметров:

```
sphereplot(r-exp, theta=диапазон, phi=диапазон)
sphereplot([r-exp,theta-expr,phi-expr], param1=диапазон, param2=диапазон)
```

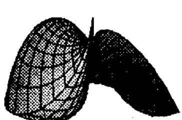
Команды примера 4.19 иллюстрируют построение поверхностей в сферической системе координат.

Пример 4.19. Построение поверхности в сферической системе координат

```
> # Единичная сфера с центром в начале координат.
> sphereplot(1,theta=0..2*Pi,phi=0..Pi,shading=ZGRAYSCALE,
            scaling=CONSTRAINED);
```



```
> # Параметрическая поверхность
> sphereplot([exp(s)+t,cos(s+t),t^2],s=0..2*Pi,t=-2..2,
            shading=ZGRAYSCALE,orientation=[80,30],grid=[40,40]);
```

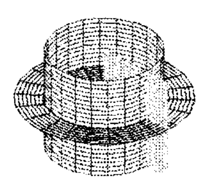


Как отмечалось при описании опций пространственных команд, Maple позволяет строить поверхности, заданные и в других пространственных системах координат. Единственное, что следует знать и хорошо представлять, — это каким образом определяется в них положение точки. Команда `coordplot3d()` визуализирует координатные поверхности всех возможных

систем координат, используемых в Maple. В примере 4.20 отображены некоторые из них.

Пример 4.20. Координатные поверхности пространственных систем координат

```
> # Цилиндрическая система координат.
> coordplot3d(cylindrical);
```



```
> # Сферическая система координат.
> coordplot3d(spherical,style=PATCH,orientation=[0,60],
             scaling=CONSTRAINED);
```



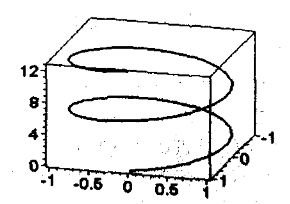
```
> # Эллиптическая система координат.
> coordplot3d(ellipsoidal,grid=[40,40],orientation=[40,60]);
```



Кривую в пространстве можно задать набором ее точек или как пересечение двух поверхностей. Команда `spacecurve()` позволяет отобразить пространственную кривую, задаваемую только набором ее точек, причем координаты точек задаются как функции одного параметра (пример 4.21).

Пример 4.21. Отображение пространственной кривой

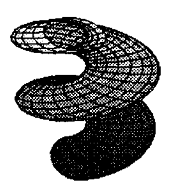
```
> spacecurve([cos(t),sin(t),t],t=0..4*Pi,color=black,thickness=2,
             axes=BOXED,orientation=[15,65]);
```



Можно построить круговую цилиндрическую поверхность заданного радиуса вдоль пространственной кривой командой tubeplot(). В примере 4.22 построена такая поверхность вдоль кривой предыдущего примера.

Пример 4.22. Круговой цилиндр вдоль пространственной кривой

```
> tubeplot([cos(t),sin(t),t],t=0..4*Pi,radius=1,tubepoints=20,
         projection=0.8,orientation=[15,65],shading=ZGREYSCALE);
```

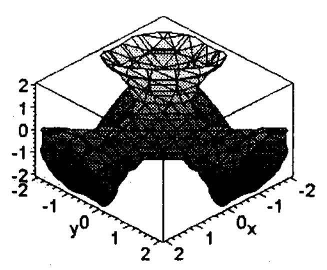


Опция radius определяет радиус криволинейного кругового цилиндра, опция tubepoints задает количество точек, используемых для построения кругового сечения цилиндра.

Для построения неявно заданных поверхностей следует использовать команду implicitplot3d(), в которой задается уравнение поверхности и диапазоны изменения всех трех ее переменных. Опцией coords можно определять построение неявно заданных поверхностей в разных системах координат.

Пример 4.23. Отображение неявно заданных поверхностей

```
> # Декартовая система координат
> implicitplot3d(x^3 + y^3 + z^3 + 1 = (x + y + z + 1)^3,x=-2..2,
              y=-2..2,z=-2..2,shading=ZGRAYSCALE,axes=BOXED);
```



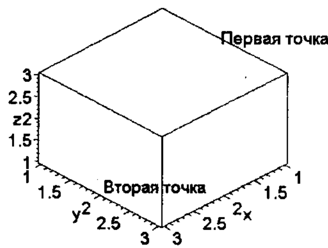

```
> # Сферическая система координат
> implicitplot3d(r^2 = (1.3)^x * sin(y),x=-1..2*Pi,y=0..Pi,r=0.1..5,
              coords=spherical,orientation=[145,95]);
```



Для создания надписей в трехмерном пространстве предназначена команда textplot3d(), синтаксис которой полностью соответствует аналогичной команде для отображения текстовых строк на плоскости с единственным исключением, связанным с заданием текстовых точек: их координаты представляются тремя числовыми значениями. Допустимая опция align выравнивания текста относительно точки имеет те же самые значения с тем же самым смыслом, что и двумерный аналог этой команды.

Пример 4.24. Отображение текста в пространстве

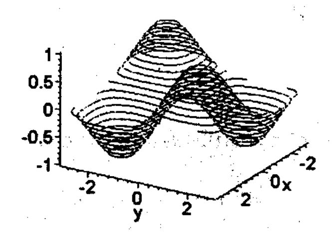
```
> textplot3d([[1,2,3,"Первая точка"],
            [3,2,1,"Вторая точка"]],axes=BOXED,color=black,
                    align={LEFT,BELOW},labels=["x","y","z"]);
```



Трехмерная команда contourplot3d() так же, как и ее двумерный аналог contourplot(), отображает линии уровня поверхности и имеет такой же синтаксис. Первое отличие заключается в том, что двумерная команда рисует линии уровня на плоскости, тогда как трехмерная отображает их в пространстве в плоскостях z=const, где постоянная const равняется значению, принимаемому функцией на соответствующей линии уровня. Второе отличие связано с реализацией: двумерная команда написана на интерпретируемом языке Maple, а трехмерная на компилируемом языке С, поэтому команда contourplot3d() выполняется быстрее команды contourplot().

Пример 4.25. Линии уровня на поверхности

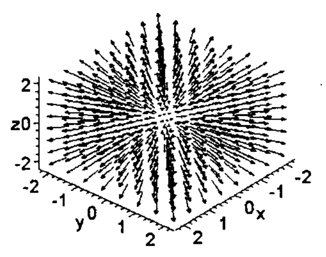
```
> contourplot3d(sin(x)*sin(y),x=-3..3,y=-3..3,
            grid=[40,40],contours=16,axes=FRAME,
            orientation=[30,60],color=black);
```



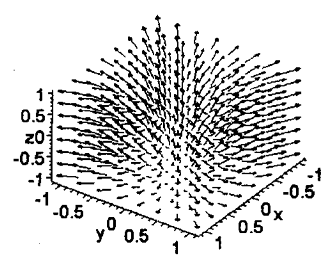
Две команды связаны с отображением векторных полей в пространстве. Команда gradplot3d() отображает поле градиента выражения, зависящего от трех переменных, в параллелепипеде, определяемом диапазонами их изменения. Команда fieldplot3d() строит в параллелепипеде, определяемом диапазонами изменения трех переменных, векторного поля с компонентами, зависящими от трех заданных переменных. В обеих этих командах можно использовать опцию arrows, значение которой определяет вид отображаемого вектора.

Пример 4.26. Построение векторных полей

```
> gradplot3d((x^2+y^2+z^2+1)^(1/2),x=-2..2,y=-2..2,z=-2..2,
        color=black,arrows=SLIM,axes=FRAME,orientation=[41,44]);
```



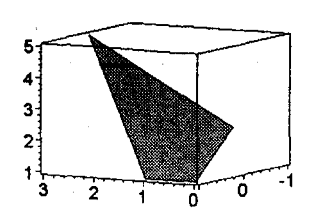

```
> fieldplot3d([2*x,2*y,1],x=-1..1,y=-1..1,z=-1..1,
        color=black,arrows=SLIM,axes=FRAME,orientation=[35,44]);
```



И последняя команда, на которой мы остановимся, это команда отображения плоского многоугольника в пространстве, который задан списком своих вершин. Для этих целей в пакет plots включена команда polygonplot3d(), синтаксис которой полностью соответствует синтаксису ее двумерного аналога polygonplot(). Единственное отличие связано с заданием точек: в трехмерной команде каждая точка представляется трехэлементным списком своих координат, причем точки не обязательно должны лежать в одной плоскости.

Пример 4.27. Отображение многоугольника в пространстве

```
> polygonplot3d([[0,1,1],[1,-1,2],[3,0,5],[1,1,1]],
        axes=boxed,orientation=[120,80]);
```



4.2.4. Трехмерные графические структуры Maple

Аналогично командам двумерной графики все команды трехмерной графики также формируют графические структуры, которые затем отображаются на выбранном устройстве отображения (по умолчанию рабочий лист или окно графики), но только в отличие от двумерных команд результатом выполнения трехмерных команд являются PLOT3D-структуры. Их так же, как и PLOT-структуры, можно распечатать командой `lprint()`:

```
> lprint(polygonplot3d([[0,1,1],[1,-1,2],[3,0,5],[1,1,1]],
                        axes=boxed,orientation=[120,80]));
PLOT3D(POLYGONS([[0., 1., 1.], [1., -1., 2.], [3., 0., 5.], [1., 1.,
1.]]),ORIENTATION(120.,80.),AXESSTYLE(BOX))
```

PLOT3D-структуры, как и PLOT-структуры, делятся на геометрические структуры, представляющие отображаемые геометрические объекты, и структуры, соответствующие опциям трехмерной графики. Все двумерные геометрические структуры и опции (см. раздел 4.1.4) используются и для формирования PLOT3D-структур с естественными изменениями в параметрах: координаты точек должны иметь три значения и в тех опциях, где необходимо задавать информацию по осям координат, следует добавлять информацию по третьей пространственной оси.

Функция `PLOT3D()` дополнительно поддерживает еще две геометрические структуры и несколько специальных трехмерных опций. Дополнительные геометрические объекты представляют поверхности с разным способом задания:

- `GRID(a..b,c..d,[z11,...,z1n],...,[zm1,...,zmn]])` — поверхности, определенные на прямоугольной области плоскости *xy* с равномерным распределением точек сетки, в которых заданы значения их *z*-координат;
- `MESH([[[x11,y11,z11],...[x1n,y1n,z1n]],...])` — поверхности, определенные координатами своих точек, причем каждая поверхность представляется списком трехэлементных списков координат точек.

В команде `PLOT3D()` можно использовать дополнительно следующие структуры-опции:

- `AMBIENTLIGHT(r,g,b)` — определяет рассеянный источник света пользовательской схемы подсветки с параметрами-числами из интервала [0,1],

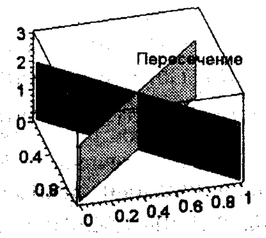
представляющие интенсивности красной, зеленой и красной его составляющих;

- COLOR(параметр) — может определять не только цвет, но и цветовую схему закрашивания поверхности, используя дополнительные возможные значения параметра: XYZSHADING, XYSHADING и ZSHADING для определения цвета объектов на основе значений координат их точек, а также ZHUE и ZGREYSCALE, являющиеся модификациями схемы ZSHADING;
- GRIDSTYLE(параметр) — задает отображаемую на поверхности сетку: треугольную (значение параметра равно TRIANGULAR) или прямоугольную (значение параметра равно RECTANGULAR);
- LIGHT(phi,theta,r,g,b) — определяет направление и интенсивность направленного источника света: первые два параметра соответствуют углам в сферической системе координат направления к источнику света и задаются в градусах, остальные три задают интенсивности красной, зеленой и синей составляющих источника света;
- LIGHTMODEL(параметр) — определяет используемую схему подсветки: пользовательскую (USER) или одну из предопределенных (LIGHT_1, LIGHT_2, LIGHT_3, LIGHT_4);
- STYLE(параметр) — определяет способ отображения поверхности и может иметь дополнительные значения параметра: HIDDEN для отображения каркаса поверхности с удалением невидимых линий, CONTOUR для отображения поверхности посредством только линий уровня и PATCHCONTOUR для отображения закрашенной поверхности с нанесенными линиями уровня.

Пример 4.28 демонстрирует непосредственное использование трехмерных графических структур для построения сложного графика в пространстве.

Пример 4.28. Отображение PLOT3D-структур

```
> # Первая плоскость зеленого цвета
> p1:=POLYGONS([[1,0,0],[1,0,2],[0,1,2],[0,1,0]],COLOR(RGB,0,1,0)):
> # Вторая плоскость красного цвета
> p2:=POLYGONS([[0,0,0],[1,1,0],[1,1,2],[0,0,2]],COLOR(RGB,1,0,0)):
> # Надпись синего цвета
> t1:=TEXT([0.5,0.5,3],"Пересечение",COLOR(RGB,0,0,1)):
> PLOT3D(p1,p2,t1,ORIENTATION(-15,45),AXES(BOXED));
```



В примере 4.28 создаются достаточно простые трехмерные графические структуры, но если необходимо создать какие-то более сложные геометрические образы, то можно воспользоваться пакетом `plottools`, содержащим ряд команд для создания трехмерных геометрических объектов, например, сферы, конуса, тора и т. д., над которыми можно даже выполнить разные преобразования с помощью команд этого же пакета. Все команды создания пространственных геометрических объектов перечислены в табл. 4.4, а перечень команд преобразования можно найти в разделе 4.1.4 данной главы. В табл. 4.4 мы не поместили команды, отмеченные звездочкой в табл. 4.2 и которые можно использовать для создания соответствующих пространственных объектов с естественным изменением синтаксиса их параметров: точки следует задавать в виде трехэлементного списка, представляющего значения трех пространственных координат.

Таблица 4.4. *Команды пакета* `plottools` *для пространственных графических структур*

Синтаксис команды	Описание создаваемой графической структуры
`cone([x,y,z],r,h,opt)`	Конус с вершиной в точке, координаты которой заданы первым параметром, направленный в положительном направлении оси *z* и высотой `h`. В сечении *z*=`h` окружность имеет радиус `r`
`cuboid([x1,y1,z1],[x2,y2,z2], opt)`	Прямоугольный параллелепипед с главной диагональю, определяемой двумя заданными точками
`cylinder([x,y,z],r,h,opt)`	Круговой цилиндр высотой `h` с образующей окружностью радиуса `r` с центром в точке, определяемой первым параметром и параллельной плоскости *xy*. Значение опции `capped`, равное `true`, отображает цилиндр с закрытыми основаниями. Если она равна `false`, то основания не закрыты
`dodecahedron([x,y,z],s,opt)`	Масштабируемый параметром `s` (по умолчанию равен `1`) додекаэдр (двенадцатигранник) с центром в точке с координатами `(x,y,z)`
`hemisphere([x,y,z],r,opt)`	Полусфера радиуса `r` с центром в точке с координатами `(x,y,z)`. Значение опции `capped`, равное `true`, отображает полусферу с закрытым сечением. Если она равна `false`, то сечение не закрыто

***Таблица 4.4** (окончание)*

Синтаксис команды	Описание создаваемой графической структуры
hexahedron([x,y,z],s,opt)	Масштабируемый параметром s (по умолчанию равен 1) шестигранник с центром в точке с координатами (x,y,z)
icosahedron([x,y,z],s,opt)	Масштабируемый параметром s (по умолчанию равен 1) икосаэдр (двадцатигранник) с центром в точке с координатами (x,y,z)
octahedron([x,y,z],s,opt)	Масштабируемый параметром s (по умолчанию равен 1) октаэдр (восьмигранник) с центром в точке с координатами (x,y,z)
semitorus([x,y,z],a..b,r,R, opt)	Часть тора с радиусом меридиана r, центром в точке с координатами (x,y,z) и радиусом образующей окружности R. Диапазон a..b определяет в радианах углы начальной и конечной точек на образующей тора. Значение опции capped, равное true, отображает часть тора с закрытыми сечениями. Если она равна false, то сечения открыты
sphere([x,y,z],r,opt)	Сфера радиуса r с центром в точке с координатами (x,y,z)
tetrahedron([x,y,z],s,opt)	Масштабируемый параметром s (по умолчанию равен 1) тетраэдр (четырехгранник) с центром в точке с координатами (x,y,z)
torus([x,y,z],r,R,opt)	Тор с радиусом меридиана r, центром в точке с координатами (x,y,z) и радиусом образующей окружности R

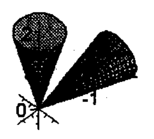
Замечание

Параметр opt во всех командах пакета plottools соответствует допустимым для соответствующей формируемой структуры опциям графической команды plot(). Некоторые команды могут иметь дополнительную опцию capped.

Пример 4.29 демонстрирует технику использования команд пакета plottools для формирования и преобразования трехмерных графических объектов.

Пример 4.29. Отображение и преобразование пространственных объектов

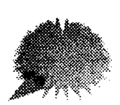
```
> # Конус и он же повернутый на угол Pi/2 относительно оси y
> PLOT3D(cone([0,0,0],1/2,2),rotate(cone([0,0,0],1/2,2), 0, Pi/2, 0),
                                AXES(NORMAL),SCALING(CONSTRAINED));
```



Вместо непосредственного использования команды PLOT3D() для отображения пространственных геометрических объектов, которая требует задания опций в виде PLOT3D-структур, можно, как и в случае с двумерной графикой, воспользоваться командой display() пакета plots, позволяющей совместить на одном графике вывод нескольких графических команд и графических структур. Пример 4.30 демонстрирует подобную технику.

Пример 4.30. Совмещение графического вывода командой display()

```
> s1:=sphere([3/2,1/4,1/2],1/4,color=red):
> s2:=sphere([3/2,-1/4,1/2],1/4,color=red):
> c:=translate(rotate(cone([0,0,0],1/2,2,color=khaki),0,Pi/2,0),3,0,1/4):
> stelhs:=stellate(rotate(hemisphere(),Pi,0,0),2):
> display(s1,s2,c,stelhs,scaling=constrained,style=patchnogrid);
```



4.3. Анимация

Графики являются превосходным способом представления информации, однако все графические команды Maple, описанию которых посвящены два предыдущих раздела, предназначены для построения статических графиков. Это означает, что если какая-либо функция зависит от некоторого параметра, то мы можем построить ряд ее графиков при различных значениях этого параметра или отобразить их одновременно на одном рисунке. Такое представление может не дать для пользователя ясную картину изменения во времени (так как именно время часто является параметром в математических моделях реальных явлений) какой-нибудь величины, представляемой в графической форме. Придется включить все свое воображение, чтобы представить изменяющуюся картину явления по его статическим "снимкам".

К счастью, разработчики Maple учли это обстоятельство и включили в пакет plots команды создания анимационной графики на плоскости и пространстве: animate() и animate3d(). Обе эти команды работают по одинаковой схеме: формируют набор графических отображений (кадров) в диапазоне изменения параметра анимации и затем последовательно отображают их друг за другом с определенной частотой точно так же, как киноаппарат последовательно проецирует на экран кадр за кадром фильм, снятый на пленку.

Для "запуска" анимации следует выделить график, отображаемый после выполнения команд анимационной графики и воспользоваться кнопками управления из контекстной панели инструментов анимации или командами меню **Animation**, расположенном в строке основного меню. Общий вид интерфейса пользователя при выделенной на рабочем листе анимационной картинки, показан на рис. 4.3. Там же можно найти описание действий, связанных с кнопками контекстной панели инструментов.

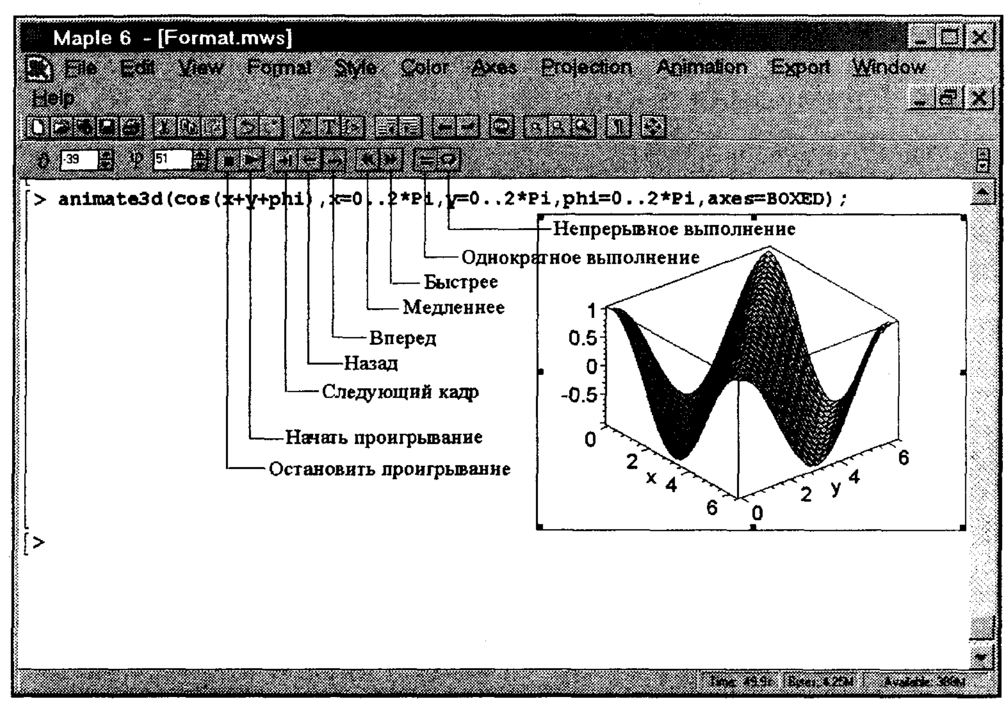


Рис. 4.3. Интерфейс пользователя для работы с анимацией

Для запуска анимации необходимо нажать кнопку **Начать проигрывание**, для остановки — **Остановить проигрывание**. Эти же действия можно выполнить и командой **Play/Stop** меню **Animation**.

Кнопка **Следующий кадр** предназначена для отображения следующего кадра анимации и позволяет последовательно просмотреть в статическом режиме все кадры, составляющие анимацию. Команда **Next** меню **Animation** выполняет те же действия.

Кнопками **Вперед** и **Назад** устанавливается режим проигрывания анимации от первого кадра к последнему и наоборот — от последнего к первому. Команда-переключатель **Forward/Backward** меню **Animation** выполняют аналогичные установки режима отображения.

Кнопки **Медленнее** и **Быстрее** соответственно уменьшают или увеличивают скорость проигрывания анимации, устанавливая отображение в диапазоне от 1 до 75 кадров в секунду. Команды **Slower** и **Faster** меню **Animation** выполняют аналогичные действия.

При нажатой кнопке **Однократное выполнение** анимация проигрывается только один раз. Нажатие кнопки **Непрерывное выполнение** осуществляет циклическое проигрывание анимации. Команда-переключатель **Single cycle/ Continuous** переключает режим отображения анимации между однократным и непрерывным.

4.3.1. Двумерная анимация

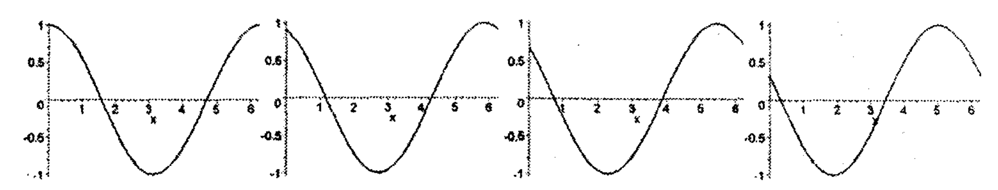
Двумерная анимация создается командой `animate()`, имеющей следующий синтаксис:

```
animate(y-expr, x=диапазон1, time=диапазон2)
```

Здесь первый параметр `y-expr` представляет выражение, зависящее от переменной `x` и параметра `time`, изменяемых в соответствующих диапазонах. В примере 4.31 создается анимационное отображение функции cos(x), которое представляет зависимость этой функции от параметра `phi`, являющегося фазой тригонометрической функции.

Пример 4.31. Двумерная анимация

```
> animate(cos(x+phi),x=0..2*Pi,phi=0..2*Pi,color=black,thickness=2);
```



Мы показали только первые четыре кадра анимационной графики, так как в книге невозможно представить динамическое отображение кадров. По умолчанию команда `animate()` создает 16 кадров. Если этого мало для плавного отображения изменения некоторой функции, то их число можно увеличить (или уменьшить), задав значение опции `frames`.

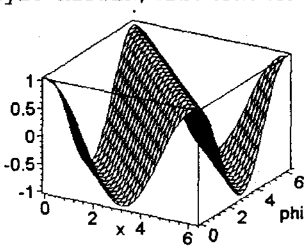
Замечание

Создание большого числа кадров анимации, особенно пространственной, может потребовать достаточно много времени и памяти компьютера.

Двумерную анимацию можно отобразить в виде пространственной статической картинки, в которой сечения по переменной-параметру и будут представлять кадры анимации (см. пример 4.32).

Пример 4.32. Пространственное представление двумерной анимации

```
> plot3d(cos(x+phi),x=0..2*Pi,phi=0..2*Pi,axes=BOXED,
                    style=HIDDEN,orientation=[-60,60],color=black);
```



Можно создавать анимацию и для параметрически заданной кривой, а также для кривой, заданной в системе координат, отличной от декартовой. Пример 4.32 демонстрирует создание анимации для параметрически заданного эллипса и для разворачивающейся спирали в полярной системе координат.

Пример 4.33. Анимации параметрической кривой и кривой в полярной системе координат

```
> animate([a*cos(t)^3,sin(t)^3,t=0..2*Pi],a=0..2);
> animate(phi*t,phi=0..8*Pi,t=1..4,coords=polar,numpoints=200);
```

Замечание

В примере 4.33 мы не стали приводить кадры анимации, а предлагаем читателю просто выполнить приведенные команды непосредственно в Maple 6.

Создать анимацию можно и командой `display()`, задав в ней список отображаемых графических структур и опцию `insequence=true`. В этом случае анимационные кадры будут составлять графические структуры из списка первого параметра команды `display()`.

Maple 6 позволяет сохранить анимацию в графическом файле формата GIF, который впоследствии можно использовать в HTML-страницах для отображения анимации, т. е. Maple 6 создает, в конечном счете, анимационный GIF-файл. Для этого следует установить графическое устройство `gif` командой `plotsetup()` и указать в ней имя файла, в котором будет сохранена анимация:

```
> plotsetup(gif, plotoutput=`d:\\plot.gif`);
> animate(cos(x+phi),x=0..2*Pi,phi=0..2*Pi,color=black,thickness=2);
```

Если вставить файл plot.gif на страницу HTML, то при ее отображении в обозревателе Интернета будет "проигрываться" анимационная картинка, отображающая изменение фазы тригонометрической функции cos(x).

4.3.2. Трехмерная анимация

В пространстве анимация создается командой `animate3d()`, имеющей синтаксис, аналогичный синтаксису команды двумерной анимации `animate()`:

```
animate3d( F(x,y,t), x=a..b, y=c..d, t=p..q, опции)
```

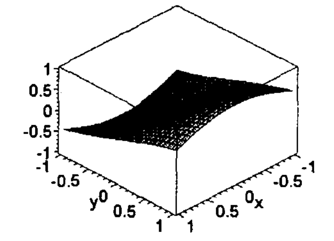
Здесь первый параметр `F(x,y,t)` представляет выражение от трех переменных или функцию двух первых независимых переменных, зависящую от параметра `t`. Остальные параметры задают диапазоны изменения независимых переменных и параметра. Опции, представленные параметром `опции`, как всегда задаются в виде уравнений и определяют вид отображаемого анимационного графика: оси, толщины линий и т. п.

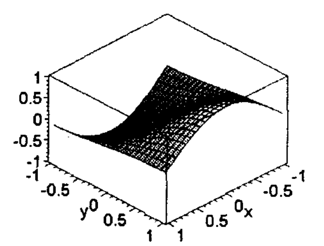
Пример 4.34. Пространственная анимация

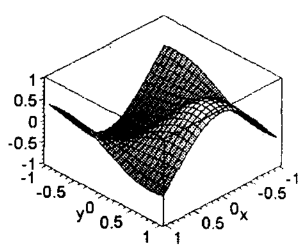
```
> F:=(x,y)->cos(t*x)*sin(t*y);
```

$$F := (x, y) \to \cos(t\,x)\sin(t\,y)$$

```
> animate3d(F(x,y),x=-1..1, y=-1..1, t=1..2,
                     axes=BOXED,shading=ZGRAYSCALE);
```







По умолчанию для представления пространственной анимации создается 8 кадров. Опцией `frame` всегда можно увеличить их число для получения более плавной картинки смены кадров. Однако следует помнить о времени и памяти, расходуемых на создание большого количества кадров анимации.

Замечание

В примере 4.34 показаны первый, четвертый и восьмой кадры анимационного изображения графика функции двух переменных.

Команда `animate3d()` может одновременно отображать изменение нескольких функций. В этом случае все они должны зависеть от одинаковых независимых переменных и одного и того же параметра и задаваться в виде списка.

Команда трехмерной анимации может создавать анимационные отображения и параметрически заданной поверхности. Пример 4.35 демонстрирует построение анимации параметрически заданной поверхности.

Пример 4.35. Пространственная анимация параметрической поверхности

```
> animate3d([x*u,t-u,x*cos(t*u)],x=1..3,t=1..4,u=2..4,
                              axes=BOXED,orientation=[-45,70]);
```

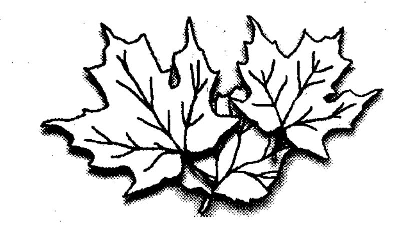
Замечание

В примере 4.35 мы не стали показывать отдельные кадры анимации. Читатель может получить представление об отображаемой поверхности, выполнив заданную в нем команду непосредственно в сессии Maple.

Аналогично двумерной анимации можно получить пространственную анимацию последовательным отображением PLOT3D-структур командой `display()`, в которой значение опции `insequence` установлено равным `true`, а отображаемые "кадры" являются элементами списка, определяемого первым параметром этой команды.

Для получения анимационного файла формата GIF следует, как и в случае двумерной анимации, вывести результаты выполнения трехмерной анимации в графическое устройство `gif`, устанавливаемое командой `plotsetup()`.

ГЛАВА 5



Основы программирования в Maple

До сих пор Maple использовался нами как интерактивная система аналитических вычислений, однако, его возможности этим не ограничиваются. Он имеет собственный язык программирования, позволяющий создавать пользовательские процедуры, на который мы будем ссылаться как на язык Maple. Вызов процедур аналогичен обращению к встроенным командам Maple, *ибо последние также представляют программы* Maple, которые, между прочим, пользователь может брать за прототипы при создании собственных. Все, что необходимо сделать при создании программы, — это написать процедуру, заключив операторы Maple между ключевыми словами `proc()` и `end proc`, и сохранить ее в библиотеке, если предполагается, что она будет необходима при решении возникающих у пользователя задач.

Отметим сразу же одно важное отличие языка Maple от других языков программирования. В нем уже реализованы многие алгоритмы, для программирования которых в других языках требуются десятки, а то и сотни строк кода, — это уже известные нам команды, такие как `solve()`, `int()` и др. При программировании в Maple в его конструкциях ветвления и цикла можно использовать весь набор доступных команд, при необходимости подключая соответствующие пакеты.

Эта глава является всего лишь введением в программирование в Maple, хотя мы постарались, пусть и бегло, осветить практически все языковые конструкции. Полностью описаны основные типы данных, все операторы управления и цикла, а также создание процедур, а вот такая конструкция, как модуль, с помощью которой можно создавать в Maple объекты и работать с ними, а также формировать новые пакеты взаимосвязанных команд, нами затронуто лишь слегка.

5.1. Язык Maple

Изучение любого языка программирования начинается с алфавита, лексем, типов данных и допустимых операций над ними. Далее описываются струк-

туры ветвления и цикла, и завершается знакомство с языком созданием процедур, объектов (если предметом изучения является объектно-ориентированный язык) и использованием средств ввода/вывода и отладки. Мы поступим аналогичным образом, и в этой главе познакомим читателя с базовыми понятиями языка Maple. Вернее, напомним, так как в гл. 2 практически мы уже познакомили с ними читателя. Здесь же известная информация будет представлена в систематизированном виде.

5.1.1. Основные элементы

Алфавит языка Maple включает 26 букв латинского алфавита, строчных (`a-z`) и прописных (`A-Z`), десять цифр (`0-9`) и 32 специальных символа, представленных в табл. 5.1.

Замечание

Напомним, что язык Maple чувствителен к регистру: строчные и прописные буквы, *используемые в именах, считаются различными.*

Таблица 5.1. *Специальные символы*

Символ	Описание	Символ	Описание
	Пробел	(	Левая круглая скобка
;	Точка с запятой	)	Правая круглая скобка
:	Двоеточие	[	Левая квадратная скобка
+	Плюс	]	Правая квадратная скобка
-	Минус или дефис	{	Левая фигурная скобка
*	Звездочка	}	Правая фигурная скобка
/	Косая черта	`	Обратный апостроф или обратные кавычки
^	"Крышка"	'	Апостроф или одинарные кавычки
!	Восклицательный знак	"	Двойные кавычки
=	Равенство	|	Вертикальная черта
<	Меньше	&	Амперсанд
>	Больше	_	Подчеркивание
@	Коммерческое AT	%	Процент
$	Знак доллара	\	Обратная косая черта
.	Точка	#	Номер
,	Запятая	?	Вопросительный знак

Из допустимых символов языка составляются *лексемы* — минимальные лексические единицы, распознаваемые интерпретатором языка Maple. К ним относятся ключевые слова (зарезервированные слова, используемые для формирования языковых конструкций), символы допустимых операций, имена переменных, функций и команд, строки, натуральные и целые числа, а также знаки препинания (разделители).

Ключевые слова имеют специальное значение для интерпретатора. Они используются для формирования конструкций ветвления и цикла, изменения принятого в соответствии с семантикой конструкции алгоритма выполнения программы, определения процедур и модулей и т. д. Эти слова нельзя использовать в качестве имен пользовательских переменных. Они являются защищенными, и любая попытка присвоить такому имени какое-либо значение приведет к ошибке выполнения. Ключевыми словами в Maple 6 являются:

```
and          end        in          od         save
break        error      intersect   option     stop
by           export     local       options    then
catch        fi         minus       or         to
description  finally    mod         proc       try
do           for        module      quit       union
done         from       next        read       use
elif         global     not         return     while
else         if
```

Над объектами Maple можно выполнять разнообразные операции: числа можно складывать, вычитать, умножать и делить; множества — пересекать, объединять, вычитать и т. д. Для операций Maple предусмотрены специальные символы и имена, которые, как и ключевые слова, можно использовать только в предписанном для них смысле. Три типа операций допустимы в Maple: унарные, бинарные и без операндов.

Унарные операции требуют для своего задания только один операнд, например, операция определения числа, противоположного заданному, — унарный минус. В табл. 5.2 представлены все унарные операции Maple.

Таблица 5.2. *Унарные операции*

Операция	Описание
+	Унарный плюс (знак числа)
-	Унарный минус (знак числа или операция смены знака содержимого переменной)

Таблица 5.2 (окончание)

Операция	Описание
!	Факториал (постфиксная операция, применяется к натуральным числам)
$	Создание последовательности (применяется к диапазону)
not	Логическое отрицание
&*string*	Формирование нейтральной операции
.	Десятичная точка в представлении вещественного числа, отделяющая целую часть от дробной (постфиксная и префиксная операция)
%*целое*	Метка

Замечание

Все операции в табл. 5.2, относительно которых не указан их тип, являются префиксными.

Бинарные операции выполняют определенные действия над двумя операндами, например, операция умножения двух чисел является бинарной, два ее сомножителя и есть два операнда, над которыми выполняется действие. Все бинарные операции перечислены в табл. 5.3.

Таблица 5.3. Бинарные операции

Операция	Описание	Операция	Описание
+	Сложение	..	Диапазон
-	Вычитание	:=	Присваивание
*	Умножение	<	Меньше чем
/	Деление	<=	Меньше или равно
^	Возведение в степень	>	Больше чем
$	Построение последовательности	>=	Больше или равно
@	Композиция функций	<>	Не равно
@@	Повторная композиция	->	Создание функции
$*string*	Нейтральная операция	union	Объединение множеств
,	Разделитель выражений	minus	Разность множеств
||	Конкатенация	intersect	Пересечение множеств
.	Десятичная точка	::	Объявление типа

Таблица 5.3 (окончание)

Операция	Описание	Операция	Описание
`and`	Логическое И	`mod`	Вычисление по модулю
`or`	Логическое ИЛИ	`.`	Некоммутативное умножение

В Maple существует три *операции без операндов*: `%`, `%%` и `%%%`, которые ссылаются, соответственно, на три предыдущих вычисленных выражения.

Все операции, за исключением нескольких, читателю знакомы по гл. 2, поэтому здесь мы остановимся только на трех пока неизвестных операциях: композиция, повторная композиция и нейтральная операция.

Операция композиция `@` применяется для создания композиции двух функций f(g(x)) (сложной функции), и ее операндами должны быть имена функций:

```
> (sin@ln)(x);
```

$$\sin(\ln(x))$$

```
> evalf((sin@ln)(Pi));
```

$$.9105984993$$

Операция `@@` используется для создания повторной композиции одной функции, определяемой ее первым операндом, например, f(f(f(x))). Вторым операндом этой операции должно быть натуральное число, задающее количество раз повторения функции в композиции:

```
> (sin@@3)(x);
```

$$(\sin^{(3)})(x)$$

```
> expand(%);
```

$$\sin(\sin(\sin(x)))$$

```
> (sin@@3)(Pi/2);
```

$$(\sin^{(2)})(1)$$

Обратите внимание, что повторная композиция отображается в области вывода как имя функции с верхним индексом, указывающим число ее повторений в композиции. Команда `expand()` раскрывает ее и представляет в виде последовательности применения функции.

Замечание

Повторную композицию не следует путать с возведением функции в степень. Например, `sin(x)^2` представляет произведение `sin(x)*sin(x)` и в области вывода отображается как $\sin(x)^2$, тогда как `(sin@@2)(x)` определяет повторную композицию функции `sin`, определяемую как `sin(sin(x))`.

Maple предоставляет пользователю возможность определить собственную операцию (унарную и бинарную), используя так называемые &-имена. Для этого после амперсанда & следует задать либо правильное имя (последовательность буквенно-цифровых символов, начинающуюся с буквы), либо последовательность специальных символов, в которой, правда, нельзя использовать символы: &, |, (,), [,], {, },;, :, ', `, #, \ и %. &-имена, или *нейтральные операции*, рассматриваются интерпретатором Maple как некоторые операции, которые не имеют никакой семантики. Они интерпретируются как вызовы функций. Если пользователь желает определить смысл нейтральной операции, то этого можно достичь, присвоив имени соответствующей нейтральной операции процедуру Maple:

```
> &~ 3;
```

&~(3)

```
> a &add c;
```

a &add *c*

```
> `&add`:=proc(x,y)
       x+y;
       end proc;
```

&add := **proc** (*x*, *y*) *x* + *y* **end proc**

```
> 2 &add b;
```

$2 + b$

Замечание

Если в нейтральной операции используются два операнда, то ее результат отображается в виде бинарной операции, если один или больше двух — то в виде неопределенной функции.

Внимание!

В сложных выражениях нейтральная операция имеет наивысший приоритет по сравнению с другими операциями. Поэтому выражение `3*2 &add 1` при семантике операции `&add`, определенной как сложение чисел, будет вычислено равным `9`, а не `7`.

Имена, которые в Maple могут быть символьными или индексными, используются для задания переменных, в которых можно сохранять результаты выполнения команд и операторов.

Символьные имена представляют собой последовательности буквенно-цифровых символов и символа подчеркивания, начинающиеся с буквы. Большое количество символьных имен защищено, и их нельзя использовать в качестве имен пользовательских переменных без снятия защиты командой `unprotect()`. Эти имена имеют определенный смысл в языке и представляют имена встроенных функций, типов данных и названия команд Maple. С другой стороны, если необходимо из каких-либо соображений защитить имя, т. е.

запретить пользователю его использование в левой части оператора присваивания, можно воспользоваться командой protect(). Снять защиту можно в любой момент командой unprotect(). Такие действия имеет смысл применять к определяемым пользователем процедурам или константам:

```
> Modulus_Steel:=2.1E5;
```

$$Modulus_Steel := 210000.$$

```
> protect('Modulus_Steel');
> Modulus_Steel:=2E5;
Error, attempting to assign to `Modulus_Steel` which is protected
```

Индексные имена представляют собой символьные имена с нижним индексом. Для их задания следует после символьного имени в квадратных скобках задать последовательность выражений, причем само индексное имя также может иметь нижний индекс:

```
> j:=1;
```

$$j := 1$$

```
> A[2,3];
```

$$A_{2,3}$$

```
> A[j^2+1,k+1][float];
```

$${A_{2,k+1}}_{float}$$

Отметим, что каждое индексное выражение в индексном имени вычисляется, т. е. если в нем встречается переменная, которой присвоено значение, то вместо нее подставляется это значение, как видно при задании последнего индексного имени в приведенном примере.

При проверке типа имени, которому не присвоено никакого значения (так называемого не вычисленного имени), команда whattype() возвращает значение symbol для символьного имени и indexed для индексного имени.

Замечание

Семантика индексного имени не предполагает, что используемое в нем символьное имя является массивом или таблицей. Так, в нашем примере A не является массивом, однако если в дальнейшем имя A будет определено как массив, то индексное имя A[2,3] будет ссылаться на соответствующий элемент массива A.

Заключение в обратные кавычки любой последовательности символов, включая пробелы и специальные символы, формирует правильное имя Maple. Дальнейшее использование подобных имен осуществляется их заданием в обратных кавычках:

```
> `n[c]~*`:=x^2+9;
```

$$n[c]\sim* := x^2+9$$

```
> `n[c]~*`;
```

$$x^2+9$$

Для создания имен можно также использовать операцию конкатенации `||` или команду `cat()`, описание которых дано в гл. 2:

```
> x || 9,x || ABC;
```

$$x9, xABC$$

```
> n:=4: x || (3*n+2);
```

$$x14$$

```
> x || (3*m+2);
```

$$x \| (3\,m+2)$$

Строка — это последовательность символов, заключенная в двойные кавычки. Ее типом является тип `string`. Для задания символа двойных кавычек (") в строке перед ним необходимо ввести символ обратной косой черты (\). Эти идущие подряд два символа трактуются как один символ двойных кавычек. Аналогично вводится и символ обратной косой черты, т. е. для задания в строке обратной косой черты его следует ввести дважды. Это связано с тем, что одиночная обратная косая черта трактуется как символ переноса длинной строки на другую строку ввода. Для выделения подстроки или символа из строки используется индексная запись.

```
> "a\"a"; length(%); # Количество символов в строке
```

"a\"a"

3

```
> "a\\a"; length(%); # Количество символов в строке
```

"a\\a"

3

```
> "This is \
> one string!";
```

"This is one string!"

```
> %[6..-1]; # Выделение подстроки, начиная с шестого символа и до конца.
```

"is one string!"

Натуральные числа — это любая последовательность цифр. Maple игнорирует стоящие слева нули. *Целые числа* — это натуральные числа или натуральные числа со знаком + или — перед ними. Следует отметить, что при проверке типа натурального числа команда `whattype()` возвращает целый тип `integer`, так как натуральный тип `natural` рассматривается как один из подтипов целого типа. Для уточнения подтипа целого числа следует использовать команду `type()`, вторым параметром которой может быть одно из следующих значений, описывающих подтипы целого типа: `natural` (натуральный), `negint` (отрицательное целое), `posint` (положительное целое), `nonnegint` (не отрица-

тельное целое), nonposint (не положительное целое), even (четное), odd (нечетное) и prime (простое).

При составлении выражений лексемы можно разделять пробельными символами (пробел, табуляция и символ новой строки). Пробельные символы не допустимы внутри лексем, их наличие приводит к ошибке синтаксического разбора выражения. Так как символ новой строки функционально идентичен пробелу, то можно вводить операторы в нескольких строках. Однако при нажатии клавиши <Enter> при незавершенном операторе интерпретатор сгенерирует ошибку. Если не обращать на нее внимание и продолжать ввод оператора, то по его завершении и передаче на выполнение оператор выполнится, как будто он был набран в одной строке.

```
> s : = 6; # Пробел в лексеме :=
Error, `=` unexpected
> a:=1+x+
>    x^2;
```

$$a := 1 + x + x^2$$

5.1.2. Выражения и типы

Выражения играют фундаментальную роль в языке Maple. В отличие от других языков программирования, в которых основным назначением выражения является его вычисление для получения числового значения, в Maple выражение не только может быть носителем числовой величины, но и представлять алгебраическое выражение с неизвестными величинами. Ведь мы помним, что система Maple, — это система аналитических вычислений, позволяющая, в том числе, производить и численные расчеты.

Выражение может состоять из числовых констант, имен переменных и неизвестных, булевых выражений, функций, рядов и других структур данных (объектов Maple), над которыми выполняются допустимые операции.

Любое выражение в Maple имеет определенный тип. Например, если выражение представляет собой сумму целых чисел, то и его тип определяется как целый (integer), если же оно представляет сумму, в слагаемые которого входят неизвестные (переменные, которым не присвоены значения) или они представляют точные числовые иррациональные величины наподобие sin(1) или sqrt(2), то такое выражение имеет тип сумма (`+`). В отличие от других языков программирования в Maple, в связи с его возможностью работать с аналитическими математическими выражениями, реализовано большое разнообразие типов. Даже такой, казалось бы, простой числовой тип, как целый (integer), соответствующий целым числам, подразделяется на отрицательный целый, положительный целый, простой и т. д. Это связано с тем, что при реализации алгоритмов аналитических вычислений существенным является не столько числовая характеристика величины, сколько

ее качественный вид: что она представляет из себя с точки зрения аналитики — сумму, произведение, целое число и т. п.

Для определения типа выражения можно использовать функцию `whattype()` с единственным аргументом, представляющим исследуемое выражение или переменную, в которой оно хранится. Правда, эта функция не определяет подтипы основного типа. Более точно определить, к какому типу принадлежит выражение можно командой `type()`, в которой первый параметр является выражением, а второй указывает допустимый тип или подтип. Эта команда возвращает `true`, если проверяемое выражение является выражением заданного типа, и `false` — в противном случае.

Каждое выражение подвергается синтаксическому разбору, результатом которого является построение *дерева выражения* (см. рис. 2.1). В первом, корневом узле отмечается тип выражения, а каждая ветвь соответствует одному из составляющих выражение членов, или операндов. Узел в каждой ветви соответствует типу операнда, так как он сам может быть сложным выражением, а его ветви определяют составляющие члены этого члена выражения. Этот процесс продолжается до тех пор, пока не дойдет до листьев дерева, представляющих имена переменных или числовые константы.

После того как дерево выражения построено, оно вычисляется. Это означает, что вместо имен переменных подставляются присвоенные им ранее значения (если таковое имело место) и вызываются и вычисляются все функции, входящие в выражение. Полученное числовое или алгебраическое выражение упрощается, а его результат и является значением выражения.

Пользователь может самостоятельно осуществить синтаксический разбор выражения, используя команды `whattype()`, `type()`, `nops()` и `op()` *(см. гл. 2)*.

В результате вычисления выражение может оказаться равным одному из основных объектов Maple. В этом случае оно будет иметь тип, имеющий соответствующий объект. В табл. 5.4 представлены основные объекты Maple и возможные типы и подтипы, которым они могут соответствовать при проверке командой `type()`.

***Таблица 5.4.** Основные объекты Maple и их типы*

Объект	Тип	Подтип
Строка	`string`	
Имя	`name`	`symbol`, `indexed`
Целое число	`integer`	`negint`, `posint`, `nonnegint`, `nonposint`, `even`, `odd`, `prime`
Дробь, или рациональное число	`rational`	`integer`, `fraction`

Таблица 5.4 (окончание)

Объект	Тип	Подтип
Десятичные числа	`numeric`	`integer, fraction, float`
	`extended_numeric`	`integer, fraction, float, infinity, undefined`
Комплексные числа	`complex(integer)` `complex(rational)` `complex(float)` `complex(number)`	
Список	`list`	
Множество	`set`	
Вызов функции (например, `g(x)`)	`function`	

При построении арифметических выражений используются арифметические операции: сложение и вычитание, умножение и деление и возведение в степень. При использовании этих операций следует знать, если все их операнды являются числами (целыми, дробными, вещественными или комплексными), то результатом вычисления такого выражения будет также некоторое число, а поэтому и тип выражения будет соответствовать типу полученного числа:

```
> s:=5+6/7+1/10+100/4*I;
```

$$s := \frac{417}{70} + 25\, I$$

```
> type(s,complex(rational));
```

true

```
> g:=expand(s^(1.+I));
```

$$g := -.8685814032 - 6.694384671\, I$$

```
> whattype(g);
```

complex

Если арифметическое выражение не вычисляется в виде числа Maple, а представляется в виде алгебраического выражения с неизвестными величинами, или при выполнении вычислений используется точная арифметика, оставляющая иррациональные значения операндов в форме точных чисел, например, `sin(1)`, `sqrt(2)` и т. д., то в этих случаях тип выражения соответствует той последней арифметической операции, которая должна быть выполнена в соответствии с приоритетом их выполнения в сложном арифметическом выражении. При этом тип алгебраической суммы, включая вычи-

тание двух операндов, определяется как `+`, тип выражения, являющегося произведением или делением некоторых операндов, будет `*`, а выражение, представляющее возведение в степень, — `^`:

```
> s:=a+1/10-100/4*I;
```

$$s := a + \frac{1}{10} - 25\,I$$

```
> whattype(s);
```

$$+$$

```
> s:=a*1/10*100/4*I;
```

$$s := \frac{5}{2}\,I\,a$$

```
> whattype(s);
```

$$*$$

```
> g:=expand(s^(1+I));
```

$$g := \left(\frac{5}{2}\,I\,a\right)^{(1+I)}$$

```
> type(g, `^`);
```

true

Следует помнить, что операндами алгебраической суммы x-y являются x и -y, операндами операции деления x/y, результат которой имеет тип произведения `*`, будут x и y^{-1}, а при возведении в степень x^a будут выделены два операнда: основание x и показатель степени a.

Разговор об операциях и типах выражений Maple будет не полным, если не затронуть операции отношения и логические операции, которые вводят алгебраический и булевый контексты.

Операции отношения <, >, <=, >=, = и <>, связывающие два алгебраических выражения, формируют новый тип выражения, семантика которого зависит от контекста, в котором оно встречается.

В *алгебраическом контексте*, который встречается при выполнении присваивания или простого задания выражения без присваивания его значения какой-либо переменной, операции отношения формируют уравнения и неравенства. Тип уравнения представляется символом `=`, а тип неравенства может быть представлен одним из символов `<>`, `<` или `<=`. При задании неравенств со знаками `>` или `>=` Maple автоматически преобразует их соответственно к неравенствам противоположного знака. Любой тип отношения имеет два операнда: левую и правую часть, которые можно выделить с помощью команд lhs() и rhs().

```
> g:=2>=x;
```

$$g := x \le 2$$

```
> whattype(g);
```

$$<=$$

```
> op(g);
```

$$x, 2$$

```
> lhs(g);
```

$$x$$

В *булевом контексте*, который возникает при использовании отношений в качестве условия в операторе `if` или после ключевого слова `while` в операторах цикла, отношение может быть равным `true` или `false`. Вычислить отношение в булевом контексте можно и командой `evalb()`, передав его ей в качестве параметра.

Если в выражении используется одна из операций отношения >, >=, < или <=, то в булевом контексте вычисляется разность левой и правой частей, которые должны быть числовыми константами, и сравнивается с нулем:

```
> x:=3:
  if x>2 then "x>2" else "x<2" end if;
```

"x>2"

```
> evalb(4+7/8+evalf(sqrt(2)) > evalf(sqrt(3))+20);
```

false

```
> evalb(4+7/8+sqrt(2) > sqrt(3)+20);
```

$$\sqrt{3}-\sqrt{2}<\frac{-121}{8}$$

```
> whattype(%);
```

$$<$$

Обратите внимание, что последнее выражение вычисляется в алгебраическом контексте, хотя явно задано его вычисление в булевом функцией `evalb()`. Это связано с тем, что обе его части не вычисляются равными числовым константам, а представляют, с точки зрения Maple, алгебраические выражения.

В случае операций = и <> их операндами могут быть произвольные выражения (алгебраические и числовые). Они проверяют, равны или не равны выражения левой и правой частей отношения с точки зрения их синтаксического представления в Maple, что не эквивалентно их равенству или не равенству с математической точки зрения.

```
> evalb(x+y=sqrt(x^2)+y);
```

true

```
> evalb(x^2-y^2=(x+y)*(x-y));
```

false

```
> evalb(x^2-y^2=expand((x+y)*(x-y)));
```

true

Второе выражение в булевом контексте, создаваемом командой `evalb()`, вычисляется как ложное равенство. Для получения правильного результата

в этом контексте пришлось явно раскрыть скобки в правой части равенства функцией expand().

Вместо команды evalb() для вычисления отношений в булевом контексте можно воспользоваться командой is(), которая проверяет равенство или неравенство не с синтаксической, а с математической точки зрения, а также в отношениях неравенства операнды могут содержать символьные числовые константы типа sqrt(2):

```
> is(x^2-y^2=(x+y)*(x-y));
```

$$true$$

```
> evalb(sqrt(3)>sqrt(2));
```

$$\sqrt{2}-\sqrt{3}<0$$

```
> is(sqrt(3)>sqrt(2));
```

$$true$$

Логические операции and, or и not также позволяют формировать новое выражение, которое вычисляется в булевом контексте, причем первые две операции являются бинарными, а последняя — префиксная унарная. Операндами логических операций могут быть выражения отношения или булевы константы true и false. Старшинство логических операций в сложном выражении следующее: сначала выполняется операция отрицания not, затем логическое умножение and и последней логическое сложение or. Скобки, как обычно, используются для изменения порядка выполнения операций в выражении в соответствии с правилом по умолчанию. Тип логического выражения соответствует последней выполняемой логической операции.

```
> 6>7 or 5>7;
```

$$false$$

```
> not a and b or c and d;
```

$$\text{not } a \text{ and } b \text{ or } c \text{ and } d$$

```
> whattype(%);
```

$$or$$

```
> not (a and (b or c) and d);
```

$$\text{not } (a \text{ and } (b \text{ or } c) \text{ and } d)$$

```
> whattype(%);
```

$$not$$

Если выражение содержит последовательность логических операций одинакового старшинства, то логические операции вычисляются последовательно слева направо до того момента, когда становится очевидным значение всего выражения. Например, при вычислении выражения, состоящего из последовательности операций логического умножения

```
> a and b and c;
```

сначала вычисляется значение выражения a. Если оно равно false, то при любых значениях выражений b и c результатом вычисления всего выражения будет false, а поэтому операнды b и c не вычисляются.

Для булева контекста необходимы булевы выражения, которые формируются с помощью булевых констант true, false и FAIL, операций отношения и логических операций. В отличие от большинства языков программирования, в которых применяется двухзначная логика со значениями true и false, Maple использует трехзначную логику, добавляя третье значение FAIL, которое можно трактовать как значение *"я не знаю"*. Команда или процедура Maple возвращает это значение, если она не может решить некоторую задачу, стоящую перед ней. Например, в случае проверки положительности некоторой переменной, которой не присвоено никакого значения:

```
> is(a, positive);
```

FAIL

Вычисления в трехзначной логике осуществляются в соответствии с таблицами истинности (табл. 5.5).

Таблица 5.5. *Таблица истинности трехзначной логики*

	Not	and			or		
		false	**true**	**FAIL**	**false**	**true**	**FAIL**
false		false	false	false	false	true	FAIL
true		false	true	FAIL	true	true	true
FAIL		false	FAIL	FAIL	FAIL	true	FAIL

Иногда необходимо провести более тонкую проверку типа выражения, чем проверка командой type() на соответствие выражения простому типу Maple. Например, проверка выражения x^2 на соответствие типу `^` может оказаться недостаточной, так как не дает пользователю никакой информации о типе основания и показателя степени. В таких случаях следует использовать *сложный*, или *структурный тип*, который формируется из простых типов Maple путем построения из них выражений, используемых в параметре задания типа команды type():

```
> type(x^2, name^integer);
```

true

После выполнения указанной проверки будет точно известно, является ли показатель степени целым числом, а основание неизвестной величиной Maple, так как тип name именно ей и соответствует:

```
> type(x^(3/2),name^integer);
```

false

```
> type((x+a)^3,name^integer);
```

false

Результат выполнения проверки на структурный тип `name^integer` двух выражений примера показывает, что ни одно из них не принадлежит к выражению указанного типа. Действительно, в первом выражении показатель степени не целый, а во втором основание не является именем переменной величины Maple. В этом последнем случае можно использовать тип `anything`, который соответствует любому выражению:

```
> type((x+a)^3,anything^integer);
```

true

Если необходимо проверить выражение на соответствие *одному* из типов из некоторого их набора, то проверяемые типы следует задавать в виде множества, причем типы могут быть совершенно разные, а подобную конструкцию можно использовать в задании структурного типа:

```
> type(2.678, {integer, float});
```

true

```
> type(2.678^x, {integer, float}^{name, float});
```

true

Для проверки типов элементов множества или списка можно воспользоваться, соответственно, структурными типами `set(тип)` и `list(тип)`. В них также в качестве параметра допускается задавать множество типов для проверки неоднородных множеств и списков.

```
> type({4,x,2.3,x^2,6.7},set({prime,name,float,name^integer}));
```

false

```
> type([1..2, "a".."g"],list(range));
```

true

Так как число `4` не является простым, то результат проверки множества `false`.

При задании структурного типа можно использовать имена встроенных или пользовательских функций для проверки, является ли выражение вызовом соответствующей функции с определенным типом параметров. При этом следует имя функции задавать в одинарных кавычках, чтобы Maple не стал вычислять такие имена как вызов соответствующих функций:

```
> type(exp(2),'exp'(integer));
```

true

```
> type(int(f(x),x=1..2),int(anything,anything));
Error, testing against an invalid type
```

В последнем примере команда проверки типа `type()` печатает сообщение о не существующем типе, так как при передаче в нее второго параметра он был вычислен как интеграл

```
> int(anything,anything);
```

$$\frac{1}{2}\,anything^2$$

и полученное значение не может быть интерпретировано как допустимый тип выражения. Для корректной проверки следует имя функции вычисления интеграла `int` в задании структурного типа заключить в одинарные кавычки:

```
> type(int(f(x),x=1..2),'int'(anything,anything));
```

true

Для проверки выражения на соответствие вызова определенной функции с произвольным числом параметров, в том числе нулевым, существует специальный тип `specfunc(тип, имя_функции)`, в котором первый параметр определяет допустимый тип параметров функции, имя которой задается вторым параметром. Проверить, что выражение является вызовом любой функции с произвольным количеством параметров заданного типа, позволяет структурный тип `function(тип)`, а тип `anyfunc(тип_1, ..., тип_n)` проверяет выражение на вызов произвольной функции с заданным количеством параметров соответствующего типа.

```
> type(diff(f(x),x),specfunc({function(anything),name},diff));
```

true

```
> type(f(1,2),function(integer));
```

true

```
> type(f(1,2,3),anyfunc({integer,float},integer));
```

false

При задании структурного типа можно использовать логические операции `And`, `Or` и `Not` (именно с прописной буквы) для формирования булевых комбинаций типов:

```
> x:=1/2*0.897; type(x,'And(constant,float)');
```

$$x := .4485000000$$

true

```
> type(Pi,'And(constant, Not(numeric))');
```

true

Внимание!

Мы рассказали о некоторых структурных типах Maple и их использовании для проверки типов выражений. Познакомиться со всеми структурными типами можно на странице справки, отображаемой командой `?type,structured`.

5.1.3. Операторы

Одним из наиболее употребительных операторов Maple является оператор присваивания :=. Его основное предназначение присвоить значение некоторой переменной, чтобы в дальнейшем использовать ее в вычислениях. Он применяется не только при создании программ, но и при интерактивной работе в Maple.

Пример 5.1. Оператор присваивания

```
> polynom3:=sum(c['i']*x^'i','i'=0..3);
```

$$polynom3 := c_0 + c_1\,x + c_2\,x^2 + c_3\,x^3$$

```
> diff_polynom3:=diff(s,x);
```

$$diff_polynom3 := c_1 + 2\,c_2\,x + 3\,c_3\,x^2$$

В примере 5.1 переменной polynom3 присваивается выражение, представляющее общий вид полинома третьей степени. В дальнейшем она используется для вычисления производной заданного полинома.

Обычно программа реализует некоторый алгоритм решения задачи. В любом алгоритме возникает необходимость выполнения определенной последовательности операторов в зависимости от истинности или ложности того или иного выражения. В Maple ветвление в программе реализуется оператором if, общая форма которого имеет следующий синтаксис:

```
if булево_выражение then последовательность_операторов
[elif булево_выражение then последовательность_операторов]
[else последовательность_операторов]
end if
```

Семантика этого оператора проста: если истинно булево выражение после ключевого слова if, то выполняется последовательность операторов после ключевого слова then до первого встретившегося elif, else или end if, если его значение равно false или FAIL, то проверяется на истинность булево выражение после ключевого слова elif, если оно задано, и в случае истинности выполняются операторы после ключевого then. Если ни одно из булевых условий не истинно, то выполняются операторы блока else, опять-таки, в случае его задания. Блоков elif может быть сколько угодно, тогда как блок else всегда только один.

Замечание

Квадратные скобки в синтаксисе операторов Maple используются нами для определения не обязательных элементов конструкций.

Пример 5.2. Оператор ветвления

```
> if type(polynom3,function(name)) then
      print("This is function call.");
  else
      print(polynom3);
  end if;
```

$$c_0 + c_1 x + c_2 x^2 + c_3 x^3$$

```
> x:=5:
> if x<0 then
      g:=-1;
  elif x<1 then
      g:=0;
  else
      g:=1;
  end if;
```

$$g := 1$$

Внимание!

В связи с использованием трехзначной логики в Maple значение `FAIL` трактуется в операторе ветвления как `false`.

Внимание!

Оператор `if` в предыдущих версиях Maple завершался ключевым словом `fi`. Для совместимости с предыдущими версиями его можно использовать и в Maple 6, однако при написании новых программ рекомендуется для завершения конструкции ветвления использовать ключевое слово `end if`.

В Maple нет оператора, реализующего конструкцию переключателя. Для этих целей следует использовать оператор `if` с несколькими блоками `elif`.

Пример 5.3. Реализация переключателя

```
> x:=5:
> if x=0 then 0
  elif x=1 then 1
  elif x=2 then 4/3
  else print("Out of range!")
  end if;
```

"Out of range!"

Синтаксис Maple позволяет использовать вложенные конструкции `if`, т. е. последовательности операторов в блоках `then` и `else` могут содержать операторы ветвления.

Операция `` `if` `` предназначена для использования в выражениях и аналогична тернарной операции условия (?) в языке С. Она имеет следующий синтаксис:

```
`if`(условие, операнд1, операнд2)
```

Семантика ее такова: вычисляется значение параметра `условие`, который должен быть булевым выражением, и если он равен `true`, то результатом операции будет `операнд1`, в противном случае `операнд2`.

Пример 5.4. Операция тернарного условия

```
> a:=4: b:=3:
> c:=`if`(a>b,a,b)+sin(`if`(a>b,a,b));
```

$$c := 4 + \sin(4)$$

Для организации повторяющихся вычислений в Maple предусмотрены две формы операторов цикла: `for-from` и `for-in`. Первый оператор цикла является универсальным и включает в себя как циклы, повторяющиеся заданное число раз, так и циклы, выполняющиеся, пока некоторое булево выражение является истинным. Вторая форма цикла `for` реализует цикл по элементам списка или множества, и в других языках программирования он известен как цикл `for-each`.

Наиболее общий синтаксис оператора цикла `for-from` следующий:

```
[for имя] [from выражение] [by выражение] [to выражение]
      [while булево_выражение]
      do последовательность_операторов end do;
```

В блоке `for` задается имя переменной цикла, блоки `from` и `to` определяют, соответственно, начальное и конечное значение диапазона изменения переменной цикла, а в блоке `by` задается шаг ее изменения (он может быть и отрицательным). Выполнение цикла начинается с присваивания переменной цикла начального значения, после чего проверяется, не превосходит ли оно конечного значения, и в случае положительного ответа выполняются операторы тела цикла, заданные в блоке `do...end do`, переменная цикла увеличивается на значение шага и алгоритм проверки начинается снова. Если значение переменной цикла превосходит конечное значение, то цикл прекращает свое выполнение. Отметим, что если при проверке начального значения переменной цикла, оно превосходит конечное значение, то цикл завершает свое выполнение, а операторы тела цикла, таким образом, не выполняются ни разу.

Если задан блок `while`, то одновременно с проверкой значения переменной цикла проверяется на истинность булево выражение этого блока, и цикл также завершает работу, если его значение оказывается равным `false` или `FAIL`.

Все перечисленные блоки являются необязательными и могут задаваться в произвольном порядке за одним исключением: если присутствует блок for, то он должен быть задан первым. Если какой-либо блок не задан, то его параметр по умолчанию принимает значение из табл. 5.6.

Таблица 5.6. *Значения по умолчанию параметров блоков цикла* for-from

Блок	Значение параметра	Блок	Значение параметра
for	Фиктивная переменная	to	infinity
from	1	while	true
by	1		

Пример 5.5. Операторы цикла

```
> # Не выполняется ни разу
> for i from 1 to -1 do evalf(sqrt(i)) end do;
> # Выполняется два раза
> for i from 1 to 2 do evalf(sqrt(i)) end do;
                              1.
                          1.414213562
> # Определяет первое простое число, большее 500000, но меньшее 500010
> for i from 5*10^5 to 500010 while not isprime(i) do end do;
> i;
                            500009
```

Последний цикл примера 5.5 демонстрирует совместное задание диапазона изменения переменной цикла и булева условия, а также тот факт, что тело цикла может быть пустым.

При задании оператора цикла for-from обязательным является только блок do, определяющий тело цикла, причем, он может быть единственным блоком цикла:

```
do последовательность_операторов end do;
```

Подобная конструкция определяет бесконечный цикл, прервать выполнение которого может только один из операторов: break, return, quit или возникновение ошибки при выполнении операторов цикла.

Если в операторе цикла отсутствуют все необязательные блоки за исключением блока while, то получается классический цикл while с предусловием: сначала проверяется истинность булева выражения условия, а затем, в зависимости от результатов проверки, либо выполняются операторы тела цикла (условие истинно), либо цикл завершает свою работу (условие ложно). При

использовании цикла while следует внимательно следить за тем, чтобы в теле цикла изменялись переменные, входящие в выражение условия, так как иначе цикл будет выполняться бесконечно.

Пример 5.6. Оператор цикла while

```
> x:=2:
> while x>1/2 do x:=x/2 end do;
```

$$x := 1$$

$$x := \frac{1}{2}$$

Вторая форма оператора цикла for-in, как уже отмечалось, организует цикл по элементам объекта, который может быть представлен последовательностью, списком, множеством, суммой, произведением или строкой. Его общий синтаксис имеет вид:

```
for имя in объект
[while булево_выражение]
do последовательность_операторов end do;
```

Переменная цикла, определяемая в блоке for...in, последовательно принимает значения операндов объекта, как они определяются командой op(). Цикл выполняется столько раз, сколько операндов задано в объекте, если только булево выражение в необязательном блоке while не станет ложным раньше, чем будут последовательно перебраны все операнды объекта.

Пример 5.7. Оператор цикла for-in

```
> # Вычисление произведения слагаемых суммы
> s:=1:
> for z in sin(x)+cos(x)
  do s:=s*z end do: s;
```

$$\sin(x)\cos(x)$$

```
> # Вычисление суммы первых 2-х сомножителей произведения
> s:=0: i:=1:
> for z  in sin(x)*sin(2*x)*sin(3*x) while i<=2
  do s:=s+z; i:=i+1 end do: s;
```

$$\sin(x) + \sin(2\,x)$$

```
> # Цикл по символам строки
> for z in "ONE" do z end do;
```

"O"

"N"

"E"

```
> # Суммирование элементов множества
> S:={x,y,z}: s:=0:
> for i in S do s:=s+i end do: s;
```

$$x + y + z$$

Обычно нормальное завершение любого цикла происходит либо когда значение переменной цикла превысило заданное конечное значение (в случае положительного шага), либо она стала меньше этого значения (в случае отрицательного шага), либо булево выражение условия в блоке `while` стало ложным. Иногда, однако, необходимо прервать дальнейшее выполнение итераций цикла при выполнении некоторого условия. Для подобных случаев в языке Maple предусмотрен оператор `break`, основное предназначение которого завершить выполнение цикла. Он используется совместно с оператором условия `if`.

Пример 5.8. Прерывание выполнения цикла оператором break

```
> # Суммирование первых двух элементов множества
> S:={x,y,z}: s:=0: j:=1:
> for i in S do
      s:=s+i;
      if j>2 then break; end if;
      j:=j+2;
  end do: s; j;
```

$$x + y$$

$$3$$

Часто приходится при программировании алгоритма применять конструкции цикла, но при некоторых условиях пропускать выполнение части или даже всех операторов тела цикла, переходя на следующий цикл с изменением значения переменной цикла. Такой эффект достигается применением оператора `next`, который немедленно прекращает выполнение текущей итерации цикла и переключает оператор цикла на выполнение очередной итерации.

Пример 5.9. Использование оператора next для перехода на очередную итерацию

```
> # Цикл только по нечетным значениям переменной цикла
> for i from 1 to 5 do
      if is(i,even) then next end if;
      i;
  end do;
```

$$1$$

$$3$$

$$5$$

Внимание!

Блок тела операторов цикла в предыдущих версиях Maple завершался ключевым словом `od`. Для совместимости с предыдущими версиями его можно использовать и в Maple 6, однако при написании новых программ рекомендуется для завершения операторов цикла использовать ключевое слово `end do`.

В отличие от других языков программирования, где для организации циклов используется подобная либо оператору `for`, либо оператору `while` конструкция, Maple предлагает программисту ряд команд, в которых реализованы часто используемые в работе циклы. Эти команды позволяют писать программы намного быстрее, освобождая программиста от "рутинного" программирования некоторых видов циклов, и, что также важно, делают их более наглядными и информативными.

К таким командам можно отнести команду `map()`, которая в цикле выполняет вызов функции, определяемой первым параметром команды, и использует в качестве первого параметра функции операнды выражения, определяемого вторым параметром команды `map()`:

```
> map(abs,x^2+sin(x));
```

$$|x|^2+|\sin(x)|$$

```
> map(int,[x^2,sin(x),f(x)],x);
```

$$\left[\frac{1}{3}x^3, -\cos(x), \int f(x)\,dx\right]$$

Если для выполнения команды необходимы дополнительные параметры, то все они задаются после второго параметра команды `map()`.

Аналогично команде `map()` работают и команды `select()` (выбрать), `remove()` (удалить) и `selectremove()` (выбрать и удалить), но их объектом могут быть только элементы списка:

```
> L:=[seq(i,i=1..10)];
```

$$L := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$$

```
> select(type,L,'odd');
```

$$[1, 3, 5, 7, 9]$$

```
> remove(type,L,'odd');
```

$$[2, 4, 6, 8, 10]$$

```
> selectremove(type,L,'odd');
```

$$[1, 3, 5, 7, 9], [2, 4, 6, 8, 10]$$

(Подробное описание этих команд можно найти в гл. 2.)

Команда `zip(f, list1, list2)` создает новый список с элементами, являющимися результатом выполнения функции двух переменных `f`, в качестве параметров которой попарно выступают элементы двух списков `list1` и

list2. Обычно создается список, длина которого равна длине наименьшего из двух заданных списков. Если при обращении к этой команде задается четвертый параметр, то длина нового списка равна длине наибольшего списка, а четвертый параметр используется в качестве недостающего параметра при вызове функции f.

```
> L1:=[seq(i,i="a".."b")];
```

$$L1 := [\text{"a"}, \text{"b"}]$$

```
> L2:=[seq(i,i=2..6)];
```

$$L2 := [2, 3, 4, 5, 6]$$

```
> zip((x,y)->x||y,L1,L2);
```

$$[\text{"a2"}, \text{"b3"}]$$

```
> zip((x,y)->x||y,L1,L2,phi);
```

$$[\text{"a2"}, \text{"b3"}, \phi 4, \phi 5, \phi 6]$$

(Подробное описание команды zip() можно найти в гл. 2.)

5.2. Процедуры

В языках программирования процедуры позволяют выделить код в одну связанную единицу с тем, чтобы в дальнейшем простым обращением к ее имени инициировать выполнение всех, содержащихся в ней операторов. Обычно процедуры используются для реализации некоторых часто выполняемых действий.

Так как основным режимом работы в Maple является интерактивное взаимодействие пользователя с его ядром, то процедуры являются единственным способом создания программ, расширяющих функциональные возможности самой системы аналитических вычислений Maple. И здесь следует отметить, что программирование в Maple достаточно своеобразно в силу аналитических "способностей" Maple. Разработка программ для выполнения численных расчетов над числовыми данными, конечно, практически не отличается от аналогичных действий в других языках программирования, разве что синтаксис используемых операторов разный, но вот как только мы хотим запрограммировать действия с аналитическими выражениями, то наш опыт численного программирования становится практически не нужным. Программирование аналитики больше связано с проверкой типов выражений, позволяющих определить, в конечном счете, их структуру и состав, а эти знания уже позволяют нам реализовать разнообразные необходимые аналитические действия над выражениями.

Небольшой пример. Пусть необходимо разработать процедуру, вычисляющую абсолютное значение некоторой величины. Программист, не работавший с аналитическими системами, но имеющий опыт программирования на

таких языках, как Fortran или C, без сомнения быстро напишет процедуру типа следующей:

```
> ABS:=proc(x)
      if x<0 then -x else x end if;
      end proc;
```

Для вещественных числовых значений параметра она будет прекрасно работать:

```
> ABS(-5.89);
```

$$5.89$$

```
> s:=1/2: ABS(s);
```

$$\frac{1}{2}$$

Но как только ей в качестве параметра будет передана переменная, которой не присвоено никакого числового значения, а такое в Maple может произойти, то наша процедура выдаст сообщение об ошибке:

```
> ABS(b);
Error, (in ABS) cannot evaluate boolean: b < 0
```

Действительно, невозможно определить, будет ли переменная `b` меньше или больше нуля. В этом случае придется немного подкорректировать процедуру с тем, чтобы она в подобных случаях возвращала не вычисленное значение нашей функции `ABS()`:

```
> ABS:=proc(x)
      if type(x,numeric) then
             if x<0 then -x else x end if;
      else
             'ABS'(x)
      end if;
      end proc:
> ABS(b);
```

$$\mathrm{ABS}(b)$$

На этом простейшем примере мы показали, какие "проблемы" возникают при программировании с аналитическими вычислениями, и показали простейший вариант решения с помощью проверки типа передаваемого в процедуру параметра. Хотя, справедливости ради, следует заметить, что на этом "проблемы" не закончились. А если передаваемое в процедуру `ABS()` выражение является произведением, например, двух неизвестных величин? Ведь нам известно, что модуль произведения равен модулю произведения модулей, но наша процедура не сделает такого преобразования, т. е. мы получим в качестве результата $\mathrm{ABS}(a\,b)$, а не $\mathrm{ABS}(a)\,\mathrm{ABS}(b)$, как следовало бы.

5.2.1. Определение процедуры

Определение процедуры Maple имеет следующий общий синтаксис:

```
proc ([ список_формальных_параметров ])
      [ local список_локальных_переменных; ]
      [ global список_глобальных_переменных; ]
      [ options список_опций; ]
      [ description строка_описания; ]
      последовательность_операторов
end proc
```

При объявлении процедуры единственным обязательным параметром является последовательность операторов, формирующих тело процедуры. Остальные параметры, определяющие локальные и глобальные переменные, список формальных параметров, задающие специальные опции режима выполнения процедуры и строку описания, могут полностью отсутствовать.

Замечание

Для совместимости с предыдущими версиями в Maple завершать объявление процедуры можно просто ключевым словом `end`. Последовательность ключевых слов `end proc` в предыдущих версиях не допустима.

Определение процедуры на рабочем листе создает объект Maple, которому, однако, следует дать имя, чтобы в дальнейшем можно было обращаться к нему. Это осуществляется обычной операцией присваивания:

```
> vector_length:=proc(x,y) sqrt(x^2+y^2); end proc;
```

$$vector_length := \mathbf{proc}(x, y)\ sqrt(x\verb|^|2 + y\verb|^|2)\ \mathbf{end\ proc}$$

После того, как процедура определена и ей присвоено имя, ее можно вызвать и выполнить с помощью оператора вызова функции, в котором задается имя процедуры, а в скобках определяются фактические переменные, которые заменяют в теле процедуры все формальные параметры:

```
> vector_length(2,5);
```

$$\sqrt{29}$$

Обычно *возвращаемым значением* процедуры является значение последнего вычисленного в теле процедуры оператора. В нашем примере последний, и единственный, оператор вычисляет квадратный корень из суммы квадратов параметров процедуры.

При вызове процедур следует помнить, что Maple сначала вычисляет все фактические параметры, а затем уже использует полученные значения в качестве параметров процедуры:

```
> z:=sqrt(y);
```

$$z := \sqrt{y}$$

```
> y:=x;
```

$$y := x$$

```
> x:=a;
```

$$x := a$$

```
> vector_length(z,y);
```

$$\sqrt{a+a^2}$$

Задать процедуру, тело которой состоит из одного выражения или одного оператора `if`, можно и с помощью специальной нотации, заимствованной из алгебры, — последовательности `'(список_параметров)->оператор'`. Определенную нами ранее процедуру можно задать и так:

```
> vector_length:=(x,y)->sqrt(x^2+y^2);
```

$$vector_length := (x, y) \to \sqrt{x^2+y^2}$$

Если в процедуре задан только один параметр, то при таком способе задания круглые скобки можно опустить:

```
> g:=x->if x<0 then -1 elif x=0 then 0 else 1 end if;
```

$g :=$ **proc**(x) **option** *operator* , *arrow*; **if** $x < 0$ **then** -1 **elif** $x = 0$ **then** 0 **else** 1 **end if end proc**

```
> g(-9),g(0),g(9);
```

$$-1, 0, 1$$

Определения процедур трактуются Maple как правильные выражения, поэтому их можно создавать и вызывать и без присваивания имен:

```
> ((x,y)->sqrt(x^2+y^2))(4,a);
```

$$\sqrt{16+a^2}$$

```
> proc(x,y) sqrt(x^2+y^2); end proc (c,d);
```

$$\sqrt{c^2+d^2}$$

Однако *неименованные процедуры* обычно используются совместно с функцией `map()`:

```
> map(x->x^3,{1,5,9,4});
```

$$\{1, 64, 125, 729\}$$

Над неименованными процедурами можно выполнять разнообразные подходящие действия, например, вычислять производную операцией `D`. Если результату, который также является процедурой, присвоить какое-либо имя, то его можно использовать для вызова полученной процедуры:

```
> F:=D((x)->sqrt(x^2+y^2));
```

$$F := x \to \frac{x}{\sqrt{x^2+y^2}}$$

```
> F(1);
```

$$\frac{1}{\sqrt{1+y^2}}$$

5.2.2. Передача параметров

При вызове процедуры Maple сначала вычисляет ее имя, а затем последовательность переданных ей параметров, причем если значением какого-либо параметра оказывается последовательность, то формируется одна результирующая последовательность величин, которая и трактуется как последовательность фактических параметров:

```
> s:=a,b; t:=2;
```

$$s := a, b$$

$$t := 2$$

```
> M:=(x,y,z)->x*y*z;
```

$$M := (x, y, z) \to x\,y\,z$$

```
> M(s,t);
```

$$2\,a\,b$$

В этом примере `x,y,z` является последовательностью формальных параметров, тогда как `a,b,2` — последовательность фактических параметров.

Количество фактических параметров не обязательно должно равняться количеству формальных параметров процедуры, определенных при ее объявлении. Если их меньше, то ошибка во время выполнения процедуры возникнет только тогда, когда при вычислении тела процедуры действительно потребуется значение этого отсутствующего параметра. Если фактических параметров больше, чем формальных, то никакой ошибки не будет сгенерировано — дополнительные параметры будут просто проигнорированы.

```
> mXYZ:=proc(x,y,z)if x>y then x else z end if end proc;
```

mXYZ := **proc** (*x*, *y*, *z*) **if** $y < x$ **then** *x* **else** *z* **end if end proc**

```
> mXYZ(1,2,3,4,5,6);
```

$$3$$

```
> mXYZ(5,1);
```

$$5$$

```
> mXYZ(1,5);
Error, (in mXYZ) mXYZ uses a 3rd argument, z, which is missing
```

Первый вызов процедуры `mXYZ()` с большим числом фактических параметров выполнился нормально. Второй вызов этой же функции с двумя параметрами тоже завершился нормально, так как значение третьего параметра не потребовалось. А вот третий вызов с двумя параметрами завершился сообщением об ошибке, что необходим третий параметр, который в вызове отсутствует.

До сих пор мы не обращали внимания на типы передаваемых в процедуру фактических параметров, а это во всех языках программирования является важной составляющей, так как позволяет исключить ошибки во время выполнения, связанные с некорректным применением некоторых операций к данным определенного типа, получаемым через параметры. Пока что наши процедуры работали без ошибок либо с параметрами любого типа, либо мы вызывали их с параметрами правильного типа, при которых подобных коллизий не возникало. Попробуем вызвать процедуру `mXYZ()` с первым параметром, являющимся алгебраической переменной:

```
> mXYZ('x',2,z);
Error, (in mXYZ) cannot evaluate boolean: -x < -2
```

Естественно, получили ошибку выполнения, так как действительно невозможно выяснить, больше или нет переменная `x` значения второго параметра. Сообщение об ошибке нам понятно, так как мы сами разрабатывали процедуру и знаем ее алгоритм, но если передать ее другому программисту, то он наверняка не поймет в чем тут дело.

Чтобы избежать подобных сообщений для процедур, работающих с определенным типом параметров, следует при объявлении процедуры объявлять и тип ее параметров. Для этих целей предназначена операция проверки типа `::`, первым операндом которой является имя формального параметра, а вторым его тип:

```
> mXYZ:=proc(x::numeric,y::numeric,z)
          if x>y then x else z end if
      end proc;
```

$mXYZ := \mathbf{proc}\ (x{::}numeric\ ,\ y{::}numeric\ ,\ z)\ \mathbf{if}\ y < x\ \mathbf{then}\ x\ \mathbf{else}\ z\ \mathbf{end\ if\ end\ proc}$

В этом случае, прежде чем начнется выполнение операторов тела процедуры, Maple проверит вычисленные значения фактических параметров соответствующим в объявлении процедуры типам формальных параметров. Если какой-нибудь параметр не будет соответствовать заданному типу, то отобразится сообщение о несоответствии параметра объявленному типу, которое более информативно, нежели сообщение о невозможности выполнить некоторую операцию:

```
> mXYZ(5,'y',z);
Error, mXYZ expects its 2nd argument, y, to be of type numeric,
but received y
```

Замечание

При проверке типа передаваемого в процедуру параметра можно использовать все основные типы Maple, структурные типы и типы, созданные пользователем.

При объявлении процедуры не обязательно задавать имена формальных параметров. Язык Maple предоставляет специальное имя `args`, с помощью ко-

торого можно получить доступ к любому переданному в процедуру фактическому параметру. Значением переменной args является последовательность всех фактических параметров, а получить значение i-го параметра можно, воспользовавшись индексной записью args[i]. Общее количество переданных при вызове процедуры фактических параметров хранится в специальной переменной nargs. Это позволяет легко писать процедуры с произвольным количеством фактических параметров (пример 5.10).

Пример 5.10. Объявление процедуры с произвольным числом параметров

```
> MAX:=proc()
       if nargs=0 then
              -infinity
       elif nargs=1 then
              args[1];
       else
              m:=args[1];
              for i from 2 to nargs do
              if args[i-1]<args[i] then args[i] else args[i-1] end if;
              end do;
       end if;
       end proc:
Warning, `m` is implicitly declared local to procedure `MAX`
Warning, `i` is implicitly declared local to procedure `MAX`
> MAX(45/46,56/57,97/99);
```

$$\frac{56}{57}$$

Процедура примера 5.10 вычисляет максимальное значение числовой последовательности любой длины. Однако при использовании подобной техники в теле процедуры следует обязательно проверять полученные параметры на соответствие допустимым типам процедуры. Понятно, что в нашем случае все фактические параметры должны быть числовыми (numeric), поэтому следующий оператор, поставленный самым первым оператором в теле процедуры, исправит это упущение:

```
for i in args do
       if not type(i,numeric) then
              error "Параметр " || i || " должен быть numeric"
       end if
end do;
```

В этом операторе цикла для завершения процедуры использован оператор error, который генерирует исключительное состояние ошибки, печатает сообщение, переданное ему в качестве параметра, и останавливает выполнение процедуры. Теперь исправленный вариант нашей процедуры при обра-

щении к ней с нечисловыми фактическими параметрами будет печатать сообщение о несоответствии параметра числовому типу:

```
> MAX(8/9,56/57,x,97/99);
Error, (in MAX) Параметр x должен быть numeric
```

В примере 5.10 после объявления процедуры `MAX()` система Maple вывела два сообщения, информирующие пользователя о том, что переменные `m` и `i`, которые используются при реализации алгоритма процедуры, неявным образом определены как ее локальные переменные. Это сообщение появилось из-за того, что мы явно не объявили эти переменные как локальные оператором `local`. Что такое локальные и глобальные переменные и как они используются при программировании в Maple, разъясняется в следующем разделе.

5.2.3. Локальные и глобальные переменные

Переменные, используемые в процедуре, могут быть либо локальными для нее, либо глобальными. Все переменные, определенные вне процедуры, являются для нее глобальными. Процедура может использовать и менять их значения, причем измененные значения будут, естественно, доступны и вне тела процедуры. Создаваемые локальные переменные доступны только в теле той процедуры, где они созданы, и их имена не конфликтуют с аналогичными именами локальных переменных других процедур и глобальными именами. Это означает, что на рабочем листе, например, могут быть определены глобальная переменная с именем `var` и много локальных переменных с аналогичным именем, но в объявлениях разных процедур. Хотя в Maple и существует алгоритм автоматического определения, какая переменная в процедуре является глобальной, а какая локальной, рекомендуется, однако, всегда явно объявлять глобальные и локальные переменные в заголовке процедуры с помощью операторов:

```
local  L1, L2, ..., Ln;
global G1, G2, ..., Gn;
```

Явно объявлять используемые переменные стоит хотя бы из-за того, чтобы не получать предупреждающих сообщений, как в примере 5.10. Измененный текст процедуры этого примера с учетом проверки типа фактических параметров окончательно выглядит так:

```
> MAX:=proc()
       local i, m
       for i in args do
              if not type(i,numeric) then
              error "Параметр " || i || " должен быть numeric"
              end if
       end do;
```

```
        if nargs=0 then
                -infinity
        elif nargs=1 then
                args[1];
        else
                m:=args[1];
                for i from 2 to nargs do
                if args[i-1]<args[i] then args[i] else args[i-1] end if;
                end do;
        end if;
        end proc:
```

Если в процедуре нет никаких объявлений локальных или глобальных переменных, то Maple пользуется следующими правилами для определения, что переменная является локальной:

- она появляется в левой части оператора присваивания;
- она появляется в качестве переменной цикла оператора `for` или как индексная переменная в командах `seq()`, `add()` или `mul()`.

Если ни одно из приведенных правил нельзя применить к переменной в теле процедуры, то она считается глобальной.

Локальные переменные также отличаются от глобальных по алгоритму их вычисления. Напомним, что для глобальных переменных используется правило полного вычисления *(см. гл. 2)*, т. е. если в выражении, которое изначально было присвоено переменной, входила неизвестная величина и впоследствии ей было присвоено какое-либо значение, то это значение подставляется в первоначальное выражение для исходной переменной. Значение, которое было присвоено этой неизвестной величине, само может содержать неизвестные величины, которым позднее также были присвоены некоторые значения. Подобная процедура выполняется и для них и повторяется до тех пор, пока не дойдет до неизвестных величин, которым не присваивалось никакого значения, или до числовых объектов Maple. Каждая такая подстановка считается одним уровнем вычисления. Локальные переменные в процедурах не вычисляются в соответствии с правилом полного вычисления. Для них Maple применяет правило вычисления точно на один уровень, т. е. если в ее определении была использована неизвестная переменная, которой впоследствии присвоили некоторое значение, то при вычислении локальной переменной этот факт не принимается во внимание. В примере 5.11 демонстрируются алгоритмы Maple вычисления глобальных и локальных переменных.

Пример 5.11. Вычисление локальной и глобальной переменной

```
> GL:=proc()
        local l,x,y;
        global g;
```

```
    g:=x^2; l:=x^2;
    x:=y^2; y:=6;
    print(g);print(l);
    end proc;
```

GL := **proc**() **local** *l*, *x*, *y*; **global** *g*; *g* := *x*^2; *l* := *x*^2; *x* := *y*^2; *y* := 6; print(*g*); print(*l*) **end proc**

```
> GL();
```

$$1296$$

$$x^2$$

```
> g;l;
```

$$1296$$

$$l$$

В примере 5.11 определена процедура `GL()`, в которой глобальной `g` и локальной `l` переменным первоначально присваиваются выражения, содержащие неизвестную величину `x`. Далее переменной `x` присваивается выражение, содержащее неизвестную величину `y`, которой чуть позже присваивается числовое значение. Завершается процедура печатью значений глобальной и локальной переменных.

Вызов этой процедуры показывает, что глобальная переменная вычислена полностью (ее значение равно числу 1296), тогда как локальная переменная только до первого уровня (ее значение равно x^2).

Вычисление значений глобальных переменных с именами `g` и `l` вне тела процедуры полностью соответствует правилам использования переменных в Maple: переменная `g` действительно имеет значение, которое она получила во время выполнения процедуры `GL()`, а переменная `l`, никоим образом не связанная с локальной переменной с таким же именем в процедуре `GL()`, не имеет никакого значения, и поэтому вычисляется равной своему имени.

Почему в теле процедуры переменные вычисляются на глубину только одного уровня, а не полностью, как в случае глобальных переменных? Это связано с проблемой эффективности выполнения процедуры, которая обычно реализует алгоритм, используя последовательное присваивание значений переменных, и практически не обращается к технике "обратного" присваивания (сначала определить переменную через неизвестную величину, которой позже присвоить некоторое значение), часто используемой при интерактивной работе. Однако если все-таки при реализации алгоритма без нее трудно обойтись, то для полного вычисления локальной переменной следует использовать команду `eval()`. Процедура `fullLoc()` примера 5.12, в которой локальная переменная вычисляется на полную глубину вложенности командой `eval(x)`, возвращает значение `4`, тогда как процедура `fullLoc1()`, возвращаемым значением которой является вычисленное на глубину одного уровня вложенности значение локальной переменной `x`, вычисляется равной y^2.

Пример 5.12. Полное вычисление локальной переменной

```
> fullLoc:=proc() local x,y; x:=y^2; y:=2; eval(x); end proc:
> fullLoc();
```

$$4$$

```
> fullLoc1:=proc() local x,y; x:=y^2; y:=2; x; end proc:
> fullLoc1();
```

$$y^2$$

В Maple V Rel 5 была введена возможность объявления и использования локальных процедур, т. е. процедур, определенных внутри другой процедуры. Для подобных процедур область действия переменных, не объявленных явно локальными или глобальными, определяется в соответствии с приведенным выше правилом для процедуры, относящейся ко всему сеансу Maple. Если переменная появляется в левой части оператора присваивания или используется в качестве переменной цикла `for`, то она считается локальной для этой процедуры и на нее распространяется правило вычисления на один уровень. Все другие переменные являются глобальными, но глобальность понимается в смысле переменных внешней процедуры, в которой объявлена локальная процедура, т. е. если во внешней процедуре есть локальная переменная с таким же именем, что и глобальная переменная во внутренней процедуре, то во внутренней процедуре используется именно она, иначе глобальная переменная внутренней процедуры является глобальной и для внешней. Пример 5.13 демонстрирует применение этого правила. В процедурах `nest()` и `nest1()` объявлены локальные процедуры `s()`, для которых переменная `z` является глобальной. Так как в процедуре `nest()` есть локальная переменная `z`, то именно она используется во вложенной процедуре. В процедуре `nest1()` переменная `z` вообще отсутствует, поэтому в ее локальной процедуре `s()` используется глобальная переменная `z`.

Пример 5.13. Вложенные процедуры

```
> nest:=proc(n) local x,s,z;
       x:=n;
       z:=nest;
       s:=proc(x) x*z end proc;
       s(x);
       end proc:
> nest1:=proc(n) local x,s;
       x:=n;
       s:=proc(x) x*z end proc;
       s(x);
       end proc:
```

```
> z:=nest1;
```

$$z := nest1$$

```
> nest(5); nest1(5);
```

$$5\ nest$$

$$5\ nest1$$

Замечание

Объявление вложенных процедур, имеющих доступ к локальным переменным внешней процедуры, в сочетании с возможностью возвращения в качестве значения внешней процедуры одной из вложенных в нее процедур можно использовать как механизм реализации инкапсуляции объектно-ориентированной технологии программирования. Если процедура возвращает вложенную в нее процедуру, которая определяет или устанавливает значение локальной переменной внешней процедуры, то такую вложенную процедуру можно рассматривать как метод некоторого объекта, свойства которого реализованы в локальных переменных внешней процедуры. Для реализации нескольких методов, однако, в этом случае приходится использовать индекс. Эта технология была единственной в версии Maple V Rel 5, реализующей работу с объектами. В версии Maple 6 для целей создания и работы с объектами следует использовать новую конструкцию `module()`, о которой немного будет рассказано в завершающем разделе данной главы.

5.2.4. Опции и строка описания

При объявлении процедуры в операторе `option` (или `options`), входящего в ее заголовок, можно определить некоторое специальное поведение процедуры, указав список допустимых опций. Для процедур используются `arrow`, `builtin`, `call_external`, `inline`, `operator`, `remember`, `system` и `trace`.

Опция `builtin` применяется для идентификации встроенных процедур Maple, находящихся в его ядре и реализующих разнообразные команды и функции. При полном вычислении подобных функций командой `eval()` опция `builtin` информирует пользователя о том, что команда является встроенной, а отображаемый уникальный номер служит для ее быстрой идентификации и выполнения:

```
> eval(map);
```

proc() option *builtin*; 189 **end proc**

Внимание!

Создать собственную встроенную процедуру нельзя.

Обычно при объявлении процедуры Maple автоматически производит некоторые упрощения в ее теле:

```
> proc(z) g:=z*z; if true then -z*z else z end if; end proc;
Warning, `g` is implicitly declared local to procedure
```

$$\textbf{proc}(z)\ \textbf{local}\ g;\ g := z\hat{}2;\ -z\hat{}2\ \textbf{end proc}$$

Сравните тело процедуры при ее объявлении с тем, как оно выглядит в области вывода, и станет ясно, какие упрощения выполняет Maple при объявлении процедуры.

Опция `operator` предписывает дальнейшие упрощения в теле процедуры, рассматривая ее как математический оператор. Два примера демонстрируют применяемые в этом случае упрощения:

```
> f1:=proc(x) h(x); end proc;
```

$$f1 := \textbf{proc}(x)\ \mathrm{h}(x)\ \textbf{end proc}$$

```
> f2:=proc(x) option operator; h(x); end proc;
```

$$f2 := h$$

Используемая совместно с опцией `operator` опция `arrow` предписывает отображать процедуру как функциональный оператор с использованием "стрелочной" нотации и эквивалентна заданию процедуры в области ввода с помощью стрелки. Следующие два объявления процедуры эквивалентны:

```
> proc(x,y) option operator, arrow; sqrt(x^2+y^2); end proc;
```

$$(x, y) \to \sqrt{x^2 + y^2}$$

```
> (x,y)->sqrt(x^2+y^2);
```

$$(x, y) \to \sqrt{x^2 + y^2}$$

Может оказаться так, что при реализации какого-то алгоритма необходимо много раз вызывать процедуру с одним и тем же набором фактических параметров. Для ускорения процесса ее вычисления она может быть сконструирована так, что результаты ее вычисления с заданными параметрами автоматически записываются в специальную *таблицу значений* функции. При последующем обращении к процедуре с этим набором фактических параметров ее значение не вычисляется вновь, а просто берется из таблицы значений. Подобная возможность существенно увеличивает скоростные характеристики процедуры, алгоритм выполнения которой достаточно сложен и может занять большой промежуток времени. Для конструирования процедуры, записывающей результаты своего вычисления с заданными значениями фактических параметров в таблицу, достаточно задать в ее теле опцию `remember`.

Использование этой опции особенно эффективно в рекурсивных процедурах. Создать рекурсивную процедуру в Maple очень просто. Если в теле процедуры есть обращение к ней же самой, то такая процедура реализуется как рекурсивная процедура. Классическим примером рекурсии является вычисление чисел Фибоначчи по формуле:

$$\mathrm{F}(n) = \mathrm{F}(n-1) + \mathrm{F}(n-2)$$

при заданных начальных условиях F(0)=0 и F(1)=1. Процедура, реализующая вычисление чисел Фибоначчи в соответствии с приведенной рекуррентной формулой, представлена в примере 5.14.

Пример 5.14. Рекурсивная процедура вычисления чисел Фибоначчи

```
> F:=proc(n::integer)
      if n<2 then
             n
      else
             F(n):=F(n-1)+F(n-2)
      end if
      end proc:
> F(300);
```

222232244629420445529739893461909967206666939096499764990979600

Эта процедура не эффективна с точки зрения ее временных характеристик. Попробуйте вычислить 2000-е число Фибоначчи и вы увидите, сколько времени потребуется ей для получения результата. Дело в том, что она каждый раз вычисляет все предыдущие числа Фибоначчи. Более эффективная реализация этого алгоритма получается с использованием опции `remember`, при задании которой в таблицу значений автоматически заносятся результаты вычислений с заданными фактическими параметрами. Теперь, если мы вычислим сразу же большое число Фибоначчи, то при вычислении меньших чисел будут использованы уже вычисленные значения из таблицы значений функции, да и вычисления с большими числами будут использовать все ранее вычисленные значения из таблицы значений. Даже первое вычисление будет более эффективно, так как некоторые значения будут браться из таблицы. Это можно увидеть, воспользовавшись командой `time()`, которая печатает затраченное процессорное время на вычисление выражения, определяемого ее параметром:

```
> time(F(20));
```

.096

```
> time(F(20));
```

.004

Первый результат получен при выполнении процедуры примера 5.14, а второй — при добавленной в объявление процедуры опции `remember`.

Занести результаты вычислений процедуры с конкретными фактическими параметрами в таблицу ее значений можно и явным образом. Для этого достаточно вызову процедуры с заданными значениями фактических параметров присвоить необходимое значение. Можно, например, реализовать алгоритм вычисления чисел Фибоначчи и способом, показанном в примере 5.15.

Пример 5.15. Явное заполнение таблицы значений

```
> F:=proc(n::integer)
      option remember;
      F(n):=F(n-1)+F(n-2)
      end proc:
> F(0):=0; F(1):=1;
```

$$F(0) := 0$$

$$F(1) := 1$$

```
> F(8);
```

$$21$$

Внимание!

Для правильной работы процедуры примера 5.15 *после* ее определения обязательно *явное* занесение в таблицу значений процедуры результатов ее вычисления при `n=0` и `n=1`.

Забегая вперед, скажем, что таблица значений процедуры представлена четвертым операндом типа `procedure`, т. е. ее можно отобразить, выполнив команду `op(4,eval(f))`, где `f` — имя процедуры. Например, после вычисления восьмого числа Фибоначчи процедурой примера 5.15 таблица ее значений выглядит следующим образом:

```
> T:=op(4,eval(F));
```

$$T := \text{table}([0 = 0, 1 = 1, 2 = 1, 3 = 2, 4 = 3, 5 = 5, 6 = 8, 7 = 13, 8 = 21])$$

Замечание

При использовании таблицы значений процедуры, особенно рекурсивной, следует помнить, что подобное действие может потребовать большого объема оперативной памяти.

Для удаления записи из таблицы значений достаточно соответствующему элементу присвоить его же собственное имя с помощью команды `evaln()`. Например, для удаления записи о пятом числе Фибоначчи из предыдущей таблицы следует воспользоваться командой:

```
> T[5]:=evaln(T[5]);
```

$$T_5 := T_5$$

```
> op(4,eval(F));
```

$$\text{table}([0 = 0, 1 = 1, 2 = 1, 3 = 2, 4 = 3, 6 = 8, 7 = 13, 8 = 21])$$

Удалить полностью всю таблицу значений процедуры можно, подставив значение `NULL` в четвертый операнд ее типа `procedure`:

```
> subsop(4=NULL,eval(F)):
> op(4,eval(F));
```

После выполнения последнего оператора никакая таблица не будет отображена, а это означает, что после применения первого оператора подстановки ее уже не существует.

Опция system используется совместно с опцией remember и идентифицирует процедуру как "системную функцию", для которой таблица значений удаляется во время процедуры сборки мусора, которая автоматически запускается системой Maple. Если для какой-либо пользовательской процедуры эта опция не указана, то при сборке мусора ее таблица значений не удаляется из памяти.

Замечание

Инициировать процедуру сборки мусора можно в любое время с помощью команды gc() (garbage collection) *без параметров.*

Любая строка в списке опций, начинающаяся со слова "Copyright", трактуется Maple как опция Copyright, которая запрещает отображать операторы тела процедуры, если только значение переменной интерфейса verboseproc, отвечающей за отображение на рабочем листе текста процедуры, не установлено равным 2 или больше:

```
> f:=proc(x,y)option `Copyright 2000`; x^2+y^2; end proc;
```

$$f := \textbf{proc}(x, y)\ \dots\ \textbf{end proc}$$

```
> interface(verboseproc=2);
> eval(f);
```

proc (*x*, *y*) **option** *`Copyright 2000`* ; *x*^2 + *y*^2 **end proc**

Последним в заголовке процедуры может быть задана строка описания в операторе description. Она не влияет на выполнение процедуры и единственное ее предназначение предоставить единственную строку комментария, которая отображается в области вывода при объявлении процедуры или ее чтения из файла, или библиотеки. Дело в том, что при распечатке процедуры из ее тела удаляются все комментарии и только эта строка может дать пользователю информацию о предназначении процедуры. Более того, если процедура определена с опцией Copyright и переменная интерфейса verboseproc имеет значение, меньшее чем 2, то эта строка является единственной отображаемой из всех операторов тела процедуры:

```
> g:=proc(x::name,y::name)
      description `Length of vector`;
      x^2+y^2; # Комментарий
      end proc;
```

g := **proc**(*x*::*name* , *y*::*name*) **description** *`Length of vector`* ; *x*^2 + *y*^2 **end proc**

```
> f:=proc(x,y)
      option `Copyright 2000`;
      description `Sum`;
```

```
    x^2+y^2;
    end proc;
```

$$f := \textbf{proc}(x, y)\ \textbf{description}\ Sum\ \ldots\ \textbf{end proc}$$

5.2.5. Возвращаемые значения

Любая процедура Maple возвращает единственное значение, которым обычно является значение последнего выполненного в теле процедуры оператора. Однако это не единственный способ передачи вычисленных в теле процедуры величин. Как и в других языках программирования, процедура Maple может возвращать значения через свои параметры, но этот способ несколько отличается от аналогичного в других языках программирования.

Для того чтобы процедура возвращала некоторое значение через параметр, необходимо в теле процедуры этому параметру присвоить требуемое значение. Процедура примера 5.16 возвращает `true`, если символ, определяемый первым параметром, содержится в строке второго параметра, в противном случае — `false`. Третий параметр возвращает позицию первого вхождения заданного символа в строку.

Пример 5.16. Возврат значений через параметр процедуры

```
> f:=proc(x::string, s::string, n::name)
    local i,j; j:=0;
    for i in s do
        j:=j+1;
        if x=i then n:=j; return true; else false; end if;
    end do;
    end proc:
> f("1","1234",n);
```

true

```
> n;
```

1

```
> f("4","1234",n);
Error, f expects its 3rd argument, n, to be of type name, but received 1
```

Процедура последовательно в цикле `for-in` проверяет равенство переданного ей символа с каждым символом строки. В случае совпадения параметру `n` присваивается номер позиции этого символа в строке и процедура завершается выполнением оператора `return`.

Первое обращение к процедуре `f()` завершается совершенно правильно, а переменная `n` содержит правильный номер позиции символа `"1"` в строке `"1234"`. Однако второй вызов этой же процедуры с параметром `n` приводит к ошибке. Дело в том, что после первого вызова переменная `n` получает зна-

чение 1 и ее тип становится равным integer, а не name, как этого требует процедура. Для выхода из этой ситуации в нашем случае следует передавать в процедуру переменную n не вычисленной, заключая ее в одинарные кавычки или выполняя команду evaln():

```
> f("4","1234",evaln(n));n;
```

true

4

```
> f("2","1234",'n');n;
```

true

2

Замечание

Процедура примера 5.16 будет корректно работать, если при каждом ее вызове в качестве третьего параметра передавать ей переменную, которой не присвоено никакого значения.

Более корректный способ исправления подобной ошибки заключается в использовании типа evaln при определении формального параметра процедуры, через который будет возвращаться некоторое значение. Этот тип Maple, используемый исключительно при задании типов параметров процедур, предписывает вычислять действительный параметр процедуры только до уровня имени. В теле процедуры его можно использовать обычным способом. Измененная процедура f() примера 5.16 представлена в примере 5.17.

Пример 5.17. Возврат значений через параметры с использованием типа evaln

```
> f:=proc(x::string,s::string, n::evaln)
       local i,j;
       j:=0;
       for i in s do
              j:=j+1;
              if x=i then n:=j; return true; else false; end if;
       end do;
       end proc:
> f("x","12x34",n);n;
```

true

3

```
> f("4","12x34",n);n;
```

true

5

Еще одним отличием использования параметров в процедурах Maple является то, что фактические параметры вычисляются только один раз при входе

в процедуру, поэтому их нельзя свободно использовать в теле процедуры, как локальные параметры: если формальному параметру в теле процедуры присвоено какое-то значение, то при дальнейшей ссылке на этот параметр мы не получим присвоенного значения. Единственное законное присваивание возможно только в случае возвращения значения через этот параметр. Процедура `simple()` примера 5.17 демонстрирует это явление. Формальный параметр `x` определен как тип `evaln(integer)`. Это означает, что соответствующий ему фактический параметр при входе в процедуру должен вычисляться до уровня имени переменной, которая хранит целое значение. В теле процедуры, так как параметр `x` является именем, чтобы использовать его значение, приходится явно прибегать к функции `eval()`. В первом операторе изменяется значение формального параметра, а сам формальный параметр является возвращаемым значением процедуры `simple()`. Казалось бы, если мы передадим в процедуру переменную со значением, например `5`, то результатом выполнения процедуры должно быть значение `6`. Однако ее вызов с указанным параметром показывает, что значением процедуры является имя `x`, а не число `6`. Здесь работает правило Maple вычисления фактических параметров один раз, а наш параметр как раз и должен вычисляться до своего имени. А вот значение самого параметра увеличилось на единицу, так как он был изменен в теле процедуры.

Пример 5.18. Использование значений параметров в теле процедуры

```
> simple:=proc(x::evaln(integer))
      x:=eval(x)+1;
      x;
      end proc:
> x:=6;
```

$$x := 6$$

```
> simple(x);
```

$$x$$

```
> x;
```

$$7$$

Правило использования фактических параметров таково: присваивать им возвращаемые значения можно только непосредственно перед выходом из процедуры. Если необходимо, например, в цикле менять значение параметра, то следует его первоначальное вычисленное значение присвоить локальной переменной, произвести вычисления, а затем присвоить формальному параметру полученный результат.

Рассмотрим процедуру `m()` примера 5.19, которая возвращает `true`, если в слагаемых первого параметра встречается переменная, заданная вторым параметром, и `false` — в противном случае. Третий параметр возвращает либо ноль, либо число слагаемых, в которых содержится эта переменная.

Пример 5.19. Неправильное использование параметров процедуры

```
> m:=proc(p::`+`,x::name,n::evaln)
        local i;
        n:=0;
        for i in p do
              if has(i,x) then n:=n+1 end if;
        end do;
        evalb(n>0);
        end proc:
> m(x+sin(x),x,n);
```

$$-n < 0$$

```
> n;
Error, too many levels of recursion
```

Она работает неправильно, так как в ней подсчет осуществляется с использованием фактического третьего параметра, который в теле процедуры всегда вычисляется до уровня своего имени, а поэтому и не работает проверка истинности логического выражения `n>0`. Более того, попытка вычислить значение третьего параметра после выполнения процедуры приводит к ошибке, так как он выражен через самого себя и это приводит к бесконечной рекурсии. Правильный вариант реализации этой процедуры с использованием локальной переменной для подсчета числа слагаемых, содержащих заданную переменную, показан в примере 5.20.

Пример 5.20. Правильное использование параметров процедуры

```
> m:=proc(p::`+`,x::name,n::evaln)
        local i,l;
        l:=0;
        for i in p do
              if has(i,x) then l:=l+1 end if;
        end do;
        n:=l;
        evalb(l>0);
        end proc:
> m(x+sin(x),x,n);
```

true

```
> n;
```

$$2$$

Прекратить выполнение процедуры и возвратить некоторое требуемое значение можно в любом месте тела процедуры. Для этих целей применяется оператор `return`. Значение следующего за ним выражения и является воз-

вращаемым значением. Мы использовали данный оператор в примере 5.16, когда определяли первое вхождение символа в заданную строку, а возвращаемым значением была позиция этого символа в строке. Общий синтаксис этого оператора следующий:

```
return выражение
```

Выражение может быть любой структурой данных, включая последовательность, список и множество. Оно может и отсутствовать, в этом случае процедура ничего не возвращает.

Кроме возврата значений процедура может возвратить ошибку, сгенерировав исключительную ситуацию, создающую специальный объект исключительного состояния, который в дальнейшем можно использовать в конструкции `try` для анализа возникшей ошибки и выполнения определенных корректирующих действий.

Возврат ошибки из процедуры осуществляется оператором `error`, который создает объект исключительного состояния, печатает сообщение об ошибке и немедленно прекращает выполнение процедуры:

```
error строка_сообщения [, список_параметры]
```

Строка сообщения — это строка, которая участвует в формировании общего сообщения об ошибке в процедуре после ее завершения оператором `error`. Если в ней заданы специальные переменные, начинающиеся с символа процента (`%`), за которым следует целое число без знака или со знаком минус, то вместо нее в строку сообщений подставляется значение переменной из списка параметров, причем абсолютное значение целого числа соответствует порядковому номеру переменной в списке. В случае целого числа со знаком минус значение соответствующего параметра преобразуется в порядковое числительное (с точки зрения английского языка) и подставляется в строку сообщений (естественно, оно должно быть целым). Пример формирования строки сообщений показан ниже:

```
> n:=6:
> error "%-1 parameter has the value %2", 2, n;
Error, 2nd parameter has the value 6
```

Специальный параметр `%0` будет отображать через запятую и пробел все значения переменных из списка параметров:

```
> error "All parameters: %0", 2,n;
Error, All parameters: 2, 6
```

Создаваемый оператором `error` объект исключительного состояния представляет последовательность выражений со следующими элементами:

- **имя процедуры, в которой было сгенерировано исключительное состояние, или константа `0`, если исключительное состояние было сгенерировано на верхнем уровне, а не в процедуре;**

- строка сообщения;
- список параметров в случае его задания.

Созданный объект исключительного состояния присваивается глобальной переменной lastexeption в виде последовательности выражений, а также для совместимости с предыдущими версиями Maple глобальной переменной lasterror присваиваются фактические параметры оператора error.

Сообщение об ошибке печатается в виде:

```
Error, (in имя_процедуры) текст_сообщения
```

Текст сообщения представляет строку сообщений с подставленными в нее значениями переменных, заданных в списке параметров оператора error. Если процедура не имеет имени, то вместо имени процедуры печатается "unknown", если ошибка сгенерирована на верхнем уровне, то часть сообщения в круглых скобках вообще не печатается, как было в наших двух предыдущих примерах выполнения оператора error.

Обычно в процедурах выход с генерацией сообщения об ошибке используется при дополнительной проверке типов параметров. Иногда стандартных и структурированных типов не достаточно для проверки передаваемых в процедуру параметров. Например, если необходимо преобразовать входной список числовых значений в список, элементами которого являются списки, состоящие из пар последовательных элементов исходного списка, то общее число элементов исходного списка должно быть четным. Осуществить такую проверку допустимыми типами Maple не представляется возможным. Одно из решений показано в примере 5.21.

Пример 5.21. Проверка входных параметров и генерация ошибки

```
> pairList:=proc(L::list(numeric))
        local i,n;
        n:=nops(L);
        if irem(n,2)=1 then
                error "Список должен иметь четное число элементов, "
                "имеет %1", n
        end if;
        [seq([L[2*i-1],L[2*i]],i=1..n/2)]
        end proc:
> pairList([1,3,2,0,3]);
Error, (in pairList) Список должен иметь четное число элементов, имеет 5
> pairList([1,3,2,0,5,6]);
```

$$[[1,3],[2,0],[5,6]]$$

В процедуре можно "поймать" ошибку с помощью конструкции try-catch, **заключив в нее блок операторов, в котором может быть сгенерирована ошибка. Она имеет следующий синтаксис:**

```
try блок_проверяемых_операторов
       catch строка_ошибки : блок_операторов
[ finally блок_завершающих_операторов ]
end try
```

Когда вычисление доходит до блока try, то начинается последовательное выполнение операторов из блока проверяемых операторов. Если в результате их выполнения не сгенерировано никакой ошибки, то выполняются операторы завершающего блока finally, если он задан, иначе управление передается оператору, следующему за оператором end try.

Если при выполнении проверяемых операторов будет сгенерирована ошибка, то их дальнейшее выполнение немедленно прекращается, а строка сообщения созданного объекта исключительного состояния сравнивается со строкой ошибки во всех операторах catch блока try. Если соответствие найдено, то выполняется соответствующий блок операторов, заданный после двоеточия (:) в соответствующем блоке catch. Если соответствия не найдено, то выполняются операторы блока finally, а исключительное состояние регенерируется вне блока try.

В блоке catch через запятую может быть задано несколько строк ошибок, тогда операторы этого блока выполняются при возникновении любой из перечисленных ошибок. Если строка ошибки вообще отсутствует, то такой блок catch "ловит" все возникающие при выполнении проверяемых операторов ошибки.

Пример 5.22 схематично демонстрирует технику создания "ловушек" для ошибки. В процедуре MethodA() генерируется объект исключительного состояния со строкой "FAIL" в случае, если значение фактического параметра больше нуля. В блоке try осуществляется ее вызов, а единственный блок catch предназначен для отлавливания ошибки "FAIL" этой процедуры и вызова другого метода. Конечно, это очень примитивный пример, но он дает представление о технике вылавливания ошибок.

Пример 5.22. Ловушка для ошибки

```
> MethodA:=proc(x::numeric)
       if x>0 then
              error "FAIL"
       else
              abs(x)
       end if;
       end proc:
> MethodB:=proc(x::numeric) -x end proc:
> x:=50:
  try
       result:=MethodA(x);
```

```
    catch "FAIL":
            result:=MethodB(x)
end try;
```

$$result := -50$$

Замечание

Если при выполнении блока `try` ошибка отловлена, то Maple не печатает никакого сообщения о ней.

Замечание

В предыдущих версиях Maple возможности организовывать ловушки для генерируемых исключительных состояний не предусматривалось. Блок `try-catch` является новым средством, реализованным в Maple 6.

В тех случаях, когда Maple не может выполнить некоторые действия с помощью своих команд из-за недостаточности информации о значениях фактических параметров или неразрешимости задачи, он просто повторяет в строке вывода вызов соответствующей команды. Говорят, что в этом случае процедура вернула не вычисленное значение. Подобная проблема может возникнуть и при разработке пользовательских программ. Например, такая ситуация встретилась нам во введении к данной главе, когда мы разрабатывали *функцию вычисления* абсолютного значения некоторой величины и передавали в нее в качестве параметра неизвестную величину, а не числовое значение. Там не работала булева операция проверки положительности параметра.

В таких случаях следует возвращать не вычисленное значение процедуры, вставляя в ее тело вызов ее же самой с формальными параметрами, но заключая ее имя или специальную переменную `procname`, содержащую имя процедуры, в одинарные кавычки. При этом следует проверять тип передаваемых параметров, при котором основной алгоритм работать не будет. Например, в процедуре `MAX(x,y)`, вычисляющей максимальное значение из двух ее параметров, следует проверить тип фактических параметров на соответствие типу `numeric` и предусмотреть возврат невычисленного значения процедуры:

```
> MAX:=proc(x,y)
    if type(x,numeric) and type(y,numeric) then
          if x>y then x else y end if;
    else
          'procname'(x,y)
    end if;
    end proc:
> MAX(1.2,8);
```

$$8$$

```
> MAX(sin(x),x);
```

$$MAX(\sin(x), x)$$

Если в этой процедуре не предусмотреть возможность возврата не вычисленного значения процедуры при нечисловых значениях параметра, то обязательно возникла бы ошибка невозможности вычисления операции булевого сравнения (см. введение в данный раздел), а также с помощью этой процедуры невозможно было бы нарисовать график функции, представляющей максимальное значение двух заданных функций. Разработанный нами вариант процедуры позволяет нарисовать такой график:

```
> plot(MAX(tan(x),2*x),x=0..1.5,color=black);
```

5.2.6. Объект процедура

Процедуры, таблицы, массивы и модули являются единственными объектами Maple, вычисление которых не подчиняется общепринятому правилу полного вычисления, — они вычисляются до уровня своих имен. Чтобы вычислить их полностью, требуется явное использование команды `eval()`:

```
> s:=proc(x) x^2 end proc:
> s;
```

$$s$$

```
> eval(s);
```

proc(*x*) *x*^2 **end proc**

Любая процедура Maple имеет тип `procedure`, который обладает, как и любой другой тип, определенным набором операндов. Всего их семь, и они представляют следующие элементы процедуры:

1. Последовательность формальных параметров.
2. Последовательность локальных переменных.
3. Последовательность опций.
4. Таблицу значений.
5. Строку описания.
6. Последовательность глобальных переменных.
7. Лексическую таблицу.

Определим процедуру m(), которую будем использовать в качестве "подопытного" экземпляра для доступа к операндам типа procedure:

```
> m:=proc(y::anything,x::name)
       local l,i;
       global f;
       option Copyright;
       description "integration";
       sum(int(1,x=0..i),i=1..4)
       end proc:
```

Чтобы увидеть все операнды этого типа, занесем в таблицу значений процедуру одно значение и тем самым создадим ее в памяти:

```
> m(0,x):=0;
```

$$m(0, x) := 0$$

Теперь наша процедура имеет все операнды, и мы можем осуществить к ним доступ.

Имя процедуры:

```
> m;
```

$$m$$

Сам объект процедура, отображаемый при полном вычислении его имени:

```
> eval(m);
```

proc (*y*::*anything* , *x*::*name*) **description** "integration" ... **end proc**

Так как в нашей процедуре задана опция Copyright, то тело процедуры не отображается, а отображается только строка описания.

Список формальных параметров:

```
> op(1,eval(m));
```

$$y::anything, x::name$$

Список локальных переменных:

```
> op(2,eval(m));
```

$$l, i$$

Список опций:

```
> op(3,eval(m));
```

$$Copyright$$

Так как в нашей процедуре задана только одна опция, то мы, естественно, списка не видим.

Таблица значений процедуры:

```
> op(4,eval(m));
```

$$\text{table}([(0,x)=0])$$

Строка описания:

```
> op(5,eval(m));
```

"integration"

Список глобальных переменных:

```
> op(6,eval(m));
```

$$f$$

Когда процедура отлажена и оттестирована, ее можно сохранить в файле с расширением `m` командой `save()`, которая позволяет сохранить любой объект Maple. Впоследствии, когда для вычислений потребуется эта процедура, ее можно прочитать в сеансе Maple командой `read()`. Например, следующая команда сохраняет процедуру `m()` этого раздела в файле с именем `m.m` в каталоге `Temp` диска `D`:

```
> save m, "d:\\Temp\\m.m";
```

Подключить данную процедуру в другом сеансе Maple можно следующей командой:

```
> read "d:\\Temp\\m.m";
> m(x^3,x);
```

10

На этом мы заканчиваем знакомство с процедурами Maple. Отметим, что хотя наша книга не является пособием по программированию в этой системе аналитических вычислений, мы сознательно достаточно подробно описали создание пользовательских процедур, так как практическое программирование в большинстве случаев заканчивается созданием процедур. Использование новых возможностей объектно-ориентированной парадигмы программирования в Maple с помощью модулей, а также создание и вызов внешних процедур, написанных на языке С, требует более глубоких знаний в области информатики. В последнем разделе данной главы мы кратко коснемся основных аспектов использования модулей и внешних процедур.

5.3. Работа с файлами

Maple — это система и язык, прежде всего предназначенные для выполнения математических преобразований и вычислений. Однако часто возникает необходимость во взаимодействии с другими системами и программами: либо результаты вычислений в Maple следует сохранить в формате, понимаемом другим приложением, либо, наоборот, получить и обработать данные,

созданные другим приложением. Иногда при разработке программ Maple возникает необходимость получить некоторую информацию от пользователя или отобразить некоторые промежуточные результаты непосредственно на рабочий лист или в файл, расположенный на диске. Для выполнения подобных действий в Maple предусмотрен большой набор команд ввода/вывода, которые можно использовать как в интерактивном режиме, так и в программах Maple.

Очень часто для взаимодействия с другими приложениями достаточно записать данные в текстовый файл в виде таблицы значений или, наоборот, прочитать подобную таблицу данных из файла, подготовленного другим приложением. В Maple эти действия проще всего выполнить с помощью команд `writedata()` и `readdata()`.

Команда `writedata()` записывает в текстовый файл данные, представленные в Maple множеством, списком, вектором, массивом или списком списков (тип `listlist`), и имеет следующий синтаксис:

```
writedata(имя_файла, данные [, формат [, процедура ] ])
```

Параметр *имя_файла* либо представляет строку, содержащую полное имя файла в соответствующей операционной системе, либо является дескриптором файла, создаваемым при его открытии командой `fopen()`. Использование имени файла эффективнее с той точки зрения, что в этом случае команда `writedata()` автоматически открывает и закрывает файл, если он существует, а также создает новый файл с указанным именем в случае его отсутствия. Если запись осуществляется в существующий файл, то он сначала очищается, а потом начинается запись в него нового содержания.

Параметры `имя_файла` и `данные` являются обязательными, тогда как два других могут быть опущены. Если данные представляют множество, список или вектор, то каждый элемент записывается в новой строке файла; если данные представлены массивом или списком списков, то каждая строка матрицы или подсписок списка записывается в файл отдельной строкой, в которой элементы отделены пробельными символами.

Параметр `формат` может иметь значение `integer`, `float` или `string` и определяет тип записываемых данных и как они преобразуются при их записи в текстовый файл. Если он опущен, то используется формат `float`, предполагающий, что данные являются числовыми, и преобразующий их в формат вещественных чисел с плавающей точкой с количеством значащих цифр в мантиссе, соответствующим текущему значению системной переменной `Digits`. Формат `integer` также работает с числовыми данными, но при записи преобразует их к целым числам, отбрасывая дробную часть. Если данные представляют строки Maple, то в этом случае следует использовать формат `string`. В случае двумерной матрицы данных или списка списков можно определить формат преобразования каждого столбца с помощью списка допустимых значений параметра `формат`.

Необязательный параметр процедура задает процедуру, которая используется для преобразования данных в выводимой структуре, если они не соответствуют указанному в параметре формат типу. Например, при заданном формате float встретилась строка символов. Если необязательная процедура не задана, то по умолчанию выводится сообщение о неправильных данных и никакого вывода в файл не происходит. Рекомендуется использовать следующую процедуру, которая без сообщения о несоответствии типа данных выводит в текстовый файл любые данные Maple:

```
proc(f, x) fprintf(f, "%a", x) end proc
```

В примере 5.23 в текстовый файл с именем D:\Temp\file.txt записывается содержимое матрицы размерности 4×2.

Пример 5.23. Запись данных в текстовый файл

```
> A:=array([ [1.0, 0.0000],
          [1.1, 4.01234567890123456789],
          [1.2, 1.9874],
          [1.3, 2.0764] ]);
```

$$A := \begin{bmatrix} 1.0 & 0. \\ 1.1 & 4.01234567890123456789 \\ 1.2 & 1.9874 \\ 1.3 & 2.0764 \end{bmatrix}$$

```
> writedata("d:\\temp\\file.txt",A,[float, float],
            proc(f, x) fprintf(f, "%a", x) end proc);
```

Для учебных целей в команде writedata() задана необязательная процедура преобразования нечисловых данных, если бы они встретились в исходной матрице. Например, эта команда отработала бы без ошибок, если элемент A[1,1] равнялся бы формуле x^2 или строке "xyz". Результатом выполнения команд примера 5.23 будет создание файла со следующим содержимым:

```
1.0          0
1.1          4.0123456789
1.2          1.9874
1.3          2.0764
```

Обратите внимание, что мантисса всех чисел не превосходит заданной по умолчанию точности Digits=10, которая использовалась при записи данных в файл, хотя некоторые числа в матрице A определены с большей мантиссой.

Команда readdata() читает табулированные данные, хранящиеся в текстовом файле. Ее синтаксис имеет следующий вид:

```
readdata(имя_файла [, формат ] [, число_столбцов])
```

Параметр `имя_файла` имеет такой же смысл, что и аналогичный параметр команды `writedata()`. Если он задан в виде полного имени файла, то этот файл автоматически открывается только для чтения и закрывается по завершении команды `readdata()`.

Необязательный параметр `формат` определяет тип хранящихся в файле данных и способ их преобразования при чтении в Maple и может иметь три значения `integer`, `float` и `string`, полностью соответствующих аналогичным значениям параметра `формат` команды `writedata()`. Значением по умолчанию является `float`.

По умолчанию команда `readdata()` читает только первый столбец данных из текстового файла, представляя его в виде списка. Если необходимо прочитать большее число столбцов, то параметр `формат` должен быть представлен списком, в котором для каждого столбца указывается формат чтения его данных. В этом случае команда представляет данные в виде списка списков, каждый подсписок которого содержит данные одной строки файла.

Задать количество читаемых столбцов можно и необязательным параметром `число_столбцов`, но при этом в параметре `формат` можно указать только один формат чтения, применяемый ко всем данным, т. е. список форматов в параметре `формат` и задаваемое третьим параметром число столбцов несовместимы.

В примере 5.24 читаются данные, записанные в файл в примере 5.23, и полученный список списков преобразуется в матрицу. Перед чтением установлено значение параметра `Digits` равным `4`.

Пример 5.24. Чтение данных из текстового файла

```
> Digits:=4:
> B:=readdata("d:\\temp\\file.txt",[float,float]);
```

$$B := [[1.0, 0.], [1.1, 4.012], [1.2, 1.987], [1.3, 2.076]]$$

```
> C:=convert(B,array);
```

$$C := \begin{bmatrix} 1.0 & 0. \\ 1.1 & 4.012 \\ 1.2 & 1.987 \\ 1.3 & 2.076 \end{bmatrix}$$

Кроме этих двух достаточно удобных и простых команд взаимодействия с внешними файлами, язык Maple предлагает большой набор команд ввода/вывода, которые по своей функциональности и своему синтаксису близки к соответствующим функциям ввода/вывода языка программирования С.

Прежде чем читать данные из файла или, наоборот, записывать в него, его следует открыть. Все команды Maple чтения/записи автоматически открывают файл при задании его имени в первом параметре. Однако можно соз-

дать *дескриптор файла* и использовать его для ссылки на соответствующий файл в командах ввода/вывода. Для этого необходимо открыть файл командой fopen() или open() и присвоить ее результат переменной, имя которой и будет дескриптором соответствующего файла.

Язык Maple поддерживает как *буферизованный*, так и *небуферизованный* ввод/вывод в файл. Если при операциях ввода/вывода данные сначала накапливаются в специальном буфере (некоторой области памяти), а затем, когда буфер полностью заполняется или файл закрывается соответствующей командой, все данные за одну операцию записываются в файл, а при чтении данные из файла сначала передаются в буфер, а потом уже читаются из него, то такой файл называется буферизованным, и Maple рассматривает его как файл типа STREAM. Если данные при операциях чтения/записи непосредственно читаются/записываются в файл, минуя промежуточный накопитель-буфер, то говорят, что осуществляется небуферизованный ввод/вывод, файл называется небуферизованным, и для Maple он имеет тип RAW. Буферизованные операции ввода/вывода выполняются быстрее небуферизованных, так как за одну операцию пересылается большее число данных.

Команда fopen() открывает буферизованные файлы и имеет синтаксис:

```
fopen(имя_файла, режим_доступа [, тип_файла])
```

Имя открываемого файла задается строкой и должно удовлетворять принятому в операционной системе именованию файлов.

Параметр режим_доступа может принимать одно из следующих значений READ, WRITE или APPEND, которые, соответственно, определяют, что файл открывается только для чтения, для чтения и записи данных с начала файла или в режиме чтения и добавления новых данных в конец существующих.

Необязательным параметром тип_файла задается текстовый (TEXT) или двоичный (BINARY) файл. Некоторые операционные системы (DOS, Windows, Macintosh) различают файлы, содержащие символы, и файлы, содержащие последовательность байтов. Первые называются текстовыми, а вторые двоичными файлами. Главное отличие при обработке таких файлов в соответствующих операционных системах заключается в различной трактовке символа новой строки, который обозначает конец одной строки и начало следующей. В некоторых операционных системах он представляется двумя последовательными байтами с ASCII-кодами 13 и 10 (DOS, Windows), в других одним с кодом 13 (Macintosh). В Maple он представляется одним символом, хотя и задается двумя "\n", и его внутренним представлением является байт со значением 10. Так вот, в случае текстового файла, когда Maple записывает в него символ новой строки, он транслируется в соответствующую последовательность байтов, которая интерпретируется операционной системой как символ новой строки, и, наоборот, при чтении эта последовательность преобразуется в байт со значением 10. При обработке двоичного файла Maple читает всю последовательность символов, трактуемую как символ новой

строки, а записывает символ новой строки как байт с кодом 10. Когда Maple выполняется в операционной системе UNIX, не различающей текстовые и двоичные файлы, он обрабатывает их как двоичные, не делая никакого преобразования символа новой строки. Если последний параметр не задан, то по умолчанию файл открывается как текстовый.

Следующая команда открывает файл в режиме чтения и создает дескриптор файла `f`, который в дальнейшем можно использовать в командах чтения/записи в файл:

```
> f:=fopen("D:\\Temp\\file.txt", APPEND);
```

$$f := 1$$

Внимание!

В качестве дескриптора файла можно использовать не только переменную, стоящую в левой части оператора присваивания (хотя это более удобно), но и просто целое значение, которое система ей присваивает, т. е. значение правой части.

Открыть небуферизованный файл можно командой

```
open(имя_файла, режим_доступа)
```

Использование параметров этой команды аналогично использованию соответствующих параметров команды `fopen()`. Небуферизованный файл Maple всегда открывает как двоичный файл. При открытии небуферизованного файла также создается дескриптор, который можно непосредственно использовать в виде соответствующего целого числа или через переменную, которой присваивается результат выполнения команды `open()`.

После того как работа с файлом завершена, его следует закрыть любой из представленных ниже команд:

```
fclose(идентификатор_файла)
close(идентификатор_файла)
```

Обе эти команды совершенно идентичны и закрывают как буферизованный, так и небуферизованный файлы. Параметр `идентификатор_файла` может быть либо именем файла, либо его дескриптором.

Для открытого файла важна его текущая позиция, которая представляет собой местоположение внутри файла, куда или откуда начинается запись или чтение байтов файла. Любая операция чтения/записи меняет текущую позицию файла на число прочитанных/записанных байтов.

Для определения или установки текущей позиции файла используется команда `filepos()` со следующим синтаксисом:

```
filepos(идентификатор_файла [, позиция])
```

Если в качестве идентификатора файла задается имя файла, который еще не открыт, то файл открывается как небуферизированный только для чтения. В случае отсутствия параметра позиция, команда возвращает текущую позицию файла. Задание этого параметра в виде целого числа устанавливает текущую позицию файла после байта с указанным номером (значение 0 соответствует началу файла). Если параметр позиция имеет значение infinity, то текущая позиция устанавливается на конец файла, а команда возвращает общее количество байтов в файле:

```
> filepos(1,infinity);
```

$$108$$

Удалить ненужный файл можно командой fremove(идентификатор_файла), причем файл не обязательно должен быть открыт. Если файла с указанным именем не существует, то Maple отображает сообщение об ошибке.

Прочитать или записать одну строку в текстовый файл можно соответственно командами readline() и writeline():

```
readline(идентификатор_файла);
writeline(идентификатор_файла, данные);
```

Обе команды открывают файл в текстовом режиме, если его идентификатор задан в виде строки, содержащей имя файла, причем первая открывает только на чтение, а вторая на запись и чтение.

При достижении конца файла команда readline() возвращает нуль, и в случае использования имени файла в качестве идентификатора закрывает файл. Программа примера 5.25 использует одну команду readline() для открытия, чтения и закрытия файла.

Пример 5.25. Открытие, чтение и закрытие файла командой readline()

```
> PrintFile:=proc(fileName::string)
            local line;
            do
                    line:=readline(fileName);
                    if line=0 then
                            break
                    else
                            printf("%s\n",line)
                    end if
            end do;
            end proc:
> PrintFile("D:\\Temp\\file.txt");
1.0          0
```

```
1.1          4.0123456789
1.2          1.9874
1.3          2.0764
```

Команда `printf()` в процедуре `FilePrint()` примера 5.25 печатает в файл по умолчанию, которым является рабочий лист, содержимое прочитанной в переменную `line` строки файла. Файл ввода/вывода по умолчанию также имеет дескриптор `default`.

Команда `writeline()` записывает в файл новую строку, содержимое которой представлено строковым значением, заданным вторым параметром. Если задано несколько строк, разделенных запятыми, то каждая из них записывается в виде новой строки файла. Следующая команда запишет в умалчиваемый файл две строки:

```
> writeline(default,"First string", "Second string");
First string
Second string
```

$$27$$

Мы описали наиболее употребительные команды для работы с файлами, которые достаточно легко использовать для чтения и записи данных. Maple содержит еще ряд полезных для программирования команд: чтение и запись определенного числа байтов `readbytes()` и `writebytes()`, чтение и запись данных в соответствии с заданным форматом `fscanf()` и `fprintf()` и другие. Со всеми этими командами читатель может ознакомиться с помощью справочной системы Maple, тем более что их использование мало отличается от соответствующих функций языка С, с которым большинство современных программистов знакомо. А так как наша книга, повторяясь еще раз, не является книгой по программированию в Maple, то на этом мы и остановимся в нашем описании системы ввода/вывода Maple и перейдем к краткому знакомству с новыми инструментами программирования в Maple 6 — модулями и вызовом функций, написанных на языке С.

5.4. Новые возможности Maple 6

Две новые возможности Maple 6 — программирование с помощью модулей и вызов внешних процедур, написанных на языке С, — значительно расширили функциональность системы аналитических вычислений Maple. Модули позволяют программировать с использованием объектно-ориентированной парадигмы, тогда как использование внешних процедур, разработанных пользователем, может ускорить выполнение некоторых задач, так как откомпилированные процедуры выполняются быстрее интерпретируемых, и даже существенно расширить класс решаемых задач.

5.4.1. Модули

Если процедуры позволяют программисту абстрагировать некоторый код, который впоследствии можно инициировать простым заданием имени процедуры, то модули предназначены для абстрагирования данных и процедур их обработки. Их главное предназначение — реализация абстрактного типа данных, т. е. данных и связанных с ними процедур их обработки, причем пользователь не имеет возможности непосредственно обращаться к самим данным, получая или изменяя их значения, а только через специальный набор процедур, который разработчик абстрактного типа данных предоставил (экспортировал) пользователю. Подобная возможность называется *инкапсуляцией* — данные как бы заключены в некую капсулу и добраться до них можно только с помощью предоставленных процедур, которые образуют так называемый интерфейс. Использование абстрактных данных при реализации больших программ имеет несомненное преимущество по сравнению с использованием традиционных сложных структурированных данных — реализация абстрактного типа скрыта от пользователя, и ее можно улучшить или даже изменить в любой момент, не меняя ничего в программе, при условии, конечно, что изменения не затрагивают интерфейса.

В Maple 6 кроме целей реализации абстрактных типов данных модули можно использовать для реализации *пакетов* — специально организованного набора связанных процедур для решения задач какой-либо определенной области знаний. Для получения доступа к процедурам пакета его необходимо подключить командой `with`.

Модули Maple являются инструментом создания *программных объектов*, обладающих свойствами и методами. Свойства объекта — это локальные переменные модуля, а методы — процедуры его интерфейса. В программе, реализованной с помощью объектов, вычисления осуществляются путем посылки объектам "сообщений" с просьбой выполнить реализованную в них функциональность (методы), на которые они отвечают выполнением затребованного сервиса.

Объявление модуля практически мало отличается от объявления процедуры:

```
module()
   [export список_экспортируемых_переменных;]
   [local список_локальных_переменных;]
   [global список_глобальных_переменных;]
   [option список_опций;]
   [description строка_описания;]
   операторы_модуля
end module
```

Оно начинается с ключевого слова `module`, за которым следуют пустые скобки, и завершается последовательностью ключевых слов `end module`, которое

может быть заменено одним ключевым словом end. Между этими ключевыми словами располагаются необязательные "описательные" операторы и последовательность операторов модуля, составляющих его тело.

В описательной части модуля, кроме уже знакомых нам по процедурам операторов объявления локальных (local) и глобальных (global) переменных, опций (option) и строки описания (description), может использоваться оператор export, объявляющий экспортируемые переменные модуля. Все перечисленные операторы, за исключением последнего, полностью соответствуют их аналогам, используемым при объявлении процедур (разве лишь модули поддерживают несколько иной набор опций), а вот оператор export имеет исключительно важное значение для конструкции модуль. В этом операторе описываются имена переменных и процедур, которые модуль экспортирует во внешнюю среду. По существу они также являются локальными, но в отличие от переменных из оператора local становятся доступными пользователю.

Чтобы получить доступ к экспортируемым именам модуля, называемым еще его членами, следует использовать специальную операцию ":-" выбора члена модуля. После "имени" модуля, соответствующего имени переменной, которой присваивается объект модуль, задается последовательность символов ":-", за которой следует экспортируемое имя. В примере 5.26 объявляется модуль и вызывается его процедура get() для получения значения локальной переменной модуля.

Пример 5.26. Модуль и его экспортируемые и локальные имена

```
> g:=module()
      export get;
      local loc;
      loc:=5;
      get:=proc()
            loc;
            end proc;
      end module;
```

$$g := \mathbf{module}()\ \mathbf{local}\ loc;\ \mathbf{export}\ get;\ \ \mathbf{end\ module}$$

```
> g:-get();
```

$$5$$

```
> g:-loc;
Error, module does not export `loc`
```

Как видите, попытка получения значения локальной переменной loc модуля оказалась не успешной, тогда как обращение к экспортируемой процедуре get(), возвращающей значение локальной переменной loc, привела к нужному результату. Более того, можно включить в модуль экспортируемую

процедуру, которая будет изменять значение этой же локальной переменной модуля, т. е. таким образом можно реализовать инкапсуляцию.

При вычислении в Maple определения модуля создается объект модуль, который можно мыслить как набор его экспортируемых имен, часто называемых членами модуля. В теле модуля экспортируемым именам, которые могут представлять переменные и процедуры, можно присваивать начальные значения и объявления процедур, в которых используются локальные переменные модуля. Каждое очередное вычисление определения модуля создает отдельный экземпляр модуля, инициирует его локальные и экспортируемые переменные, а также создает отдельный набор локальных переменных. В примере 5.27 определение модуля задано в процедуре `gen()`, вызов которой создает новый экземпляр модуля. Экспортируемым именем является единственная локальная переменная `e` модуля. В каждом экземпляре модуля она своя, поэтому логическая операция сравнения одной и той же локальной переменной двух экземпляров модуля завершается вычислением `false`.

Пример 5.27. Сравнение одинаковых локальных имен двух экземпляров модуля

```
> f:=proc()
      module() export e; end;
      end;
```

f := **proc() module() export** e; **end module end proc**

```
> g:=f();
```

g := **module() export** e; **end module**

```
> l:=f();
```

l := **module() export** e; **end module**

```
> g:-e;
```

e

```
> l:-e;
```

e

```
> evalb(g:-e = l:-e);
```

false

```
> evalb(e = l:-e);
```

false

Экспортируемая модулем локальная переменная не то же самое, что и глобальная переменная с таким же именем (см. последний оператор примера 5.27).

Замечание

Модули и процедуры можно свободно вкладывать друг в друга. Глубина уровня вложенности может быть произвольной.

Модуль отличается от процедуры тем, что, во-первых, он не содержит формальных параметров, а во-вторых, вычисление объявления модуля одновременно "инициирует" его, создавая набор локальных и экспортируемых переменных. Вычисление объявления процедуры только "создает" в памяти "описание" процедуры, а все ее локальные переменные создаются и вычисляются каждый раз при ее вызове. Правда, с помощью процедур можно также организовать доступ к локальным переменным, используя возвращаемую процедурой процедуру (см. пример 5.28).

Пример 5.28. Доступ к локальной переменной процедуры

```
> g:=proc()
      local loc,get;
      loc:=5;
      get:=proc()
            loc;
            end;
      end proc;
```

$$g := \mathbf{proc}()\ \mathbf{local}\ loc, get;\ loc := 5;\ get := \mathbf{proc}()\ loc\ \mathbf{end\ proc\ end\ proc}$$

```
> f:=g();
```

$$f := \mathbf{proc}()\ loc\ \mathbf{end\ proc}$$

```
> f();
```

$$5$$

В процедуре примера 5.28 используется небольшой "трюк". Локальная переменная `loc` вычисляется при инициировании вызова процедуры `g()`, возвращаемым значением которой является локальная процедура `get()`, которая, в свою очередь, возвращает значение локальной переменной `loc`. Но так как после вызова процедуры `g()` ее локальная переменная `get()` становится видимой и вне тела процедуры, то становится видимой и ее локальная переменная `loc`.

Для определения экспортируемых имен модуля предназначена команда `exports()`, единственным параметром которой является имя объекта модуль:

```
> exports(g);
```

$$get$$

В приведенной команде определяются экспортируемые имена модуля `g` примера 5.26.

Для проверки, является ли некоторое имя членом, т. е. экспортируемым именем, некоторого модуля можно использовать уже известную нам функцию `member()`:

```
> member(get,g);
```

$$true$$

```
> member(loc,g);
```

false

В объявлении модуля нет никаких формальных параметров, так как модуль, в отличие от процедуры, не "вызывается", а поэтому и нет необходимости в передаче фактических параметров. Однако каждое объявление модуля имеет неявный параметр `thismodule`, который аналогичен параметру `procname` процедуры, но отличается от последнего. Параметр `procname` вычисляется как имя процедуры, тогда как параметр `thismodule` вычисляется как сам модуль. Это позволяет создавать разные имена для одного и того же объекта, экспортируя процедуру, возвращающую сам модуль (пример 5.29).

Пример 5.29. Алиасные имена модуля

```
> m := module()
      export sameModule,getColor, setColor;
      local Color;
      Color:="Blue";
      getColor:=proc()
            Color
            end proc;
      setColor:=proc(x::string)
            Color:=x;
            end proc;
      sameModule := proc()
      thismodule
      end proc;
      end module:
> m:-getColor();m:-setColor("Red");m:-getColor();
```

"Blue"

"Red"

"Red"

```
> g:=m:-sameModule();
```

$g :=$ **module()** **local** *Color*; **export** *sameModule* , *getColor* , *setColor*; **end module**

```
> g:-getColor();
```

"Red"

```
> m:-setColor("Yellow");g:-getColor();
```

"Yellow"

"Yellow"

```
> g:-setColor("Black");m:-getColor();
```

"Black"

"Black"

В примере 5.29 создан модуль m, члены которого getColor() и setColor() соответственно читают и устанавливают значение локальной переменной Color, а член sameModule() возвращает объявление самого модуля. Если переменной g присвоить результат выполнения процедуры sameModule(), то ссылаться на первоначально созданный объект модуль с именем m можно и по имени g, причем любые изменения локальных переменных с помощью методов объекта m влияют, естественно, и на значения локальных переменных объекта g, и наоборот. Это как раз и подтверждает тот факт, что оба имени ссылаются на один и тот же объект.

Maple распознает модуль как тип с именем module. Так как само слово module является ключевым словом, то в команде проверки типа его следует заключать в обратные кавычки `module`:

```
> g:=module() export l; l:=0; end module:
> type(g,`module`);
                              true
> type(LinearAlgebra,`module`);
                              true
> type(linalg,`module`);
                              false
```

Обратите внимание, что новый пакет линейной алгебры LinearAlgebra реализован в виде модуля, тогда как старый пакет linalg модулем не является.

Завершая краткий разговор о модулях Maple, мы приведем простой пример, демонстрирующий их применение. Очень полезным при реализации разнообразных динамических структур данных является объект, который хранит пару связанных значений. Именно его мы и попробуем реализовать.

Прежде чем создавать любой объект, его следует "спроектировать", т. е. определить его состояние и поведение. Состояние программного объекта реализуется в локальных переменных модуля, а поведение, или что может "делать" объект, задается экспортируемыми процедурами. В нашем случае связанной пары некоторых значений объект должен, естественно, хранить эти значения (мы их реализуем в локальных переменных first и second), позволять получать их, а также допускать их независимое изменение. Поведение объекта представим четырьмя экспортируемыми процедурами: первые две (getFirst() и setFirst()) будут, соответственно, возвращать и устанавливать первое значение связанной пары, а вторые две (getSecond() и setSecond()) выполнять аналогичные действия для второго значения связанной пары.

Для создания объекта необходим конструктор, представляющий процедуру, которая возвращает объявление модуля, реализующего объект. Параметрами процедуры должны быть первое и второе значения связанной пары. Реализация конструктора представлена в примере 5.30.

Пример 5.30. Конструктор объекта связанной пары

```
> MakePair := proc( a, b)
            module()
            local first, second;
            export getFirst, getSecond, setFirst, setSecond;
            first := a;
            second := b;
            getFirst := () -> first;
            getSecond := () -> second;
            setFirst := proc( v) first := v end;
            setSecond := proc( v) second := v end;
            end module
            end proc:
```

Теперь можно создать объект, представляющий связанную пару значений, с помощью разработанного конструктора и проверить правильность работы экспортируемых процедур:

```
> P1:=MakePair(x,y):
> P1:-getFirst(),P1:-getSecond();
```

$$\mu, y$$

```
> P1:-setFirst(mu); P1:-getFirst(),P1:-getSecond();
```

$$\mu$$

$$\mu, y$$

Однако постоянно вызывать методы созданного объекта P1 с использованием операции ссылки на члены модуля :- не достаточно эффективно с точки зрения читаемости программы. Изменить эту ситуацию можно, создав общие процедуры вызова методов объектов. Этим процедурам в качестве параметров передаются произвольные объекты, обладающие определенными методами, что и дало им название общих, а сами методы вызываются в теле соответствующих процедур. Общие процедуры для вызова соответствующих методов нашего объекта представлены в примере 5.31.

Пример 5.31. Общие процедуры вызова методов объекта связанной пары

```
> first := proc( obj::`module`)
      if member( 'getFirst', obj) then
            obj:-getFirst()
      else
            error "Объект не поддерживает метод: get%1", procname
      end if
      end proc:
 second := proc( obj::`module`)
```

```
	if member( 'getSecond', obj) then
		obj:-getSecond()
	else
		error "Объект не поддерживает метод: get%1", procname
	end if
	end proc:
 setfirst := proc( obj::`module`, v)
	if member( 'setFirst', obj) then
		obj:-setFirst(v)
	else
		error "Объект не поддерживает метод: %1", procname
	end if
	end proc:
 setsecond := proc( obj::`module`, v)
	if member( 'setSecond', obj) then
		obj:-setSecond(v)
	else
		error "Объект не поддерживает метод: %1", procname
	end if
	end proc:
```

Теперь вызов методов объекта P1 выглядит "лучше", чем с использованием операции `:-`:

```
> first(P1);
```

$$\mu$$

```
> setsecond(P1,lambda);
```

$$\lambda$$

```
> first(P1),second(P1);
```

$$\mu, \lambda$$

```
> first(1);
Error, (in first) Объект не поддерживает метод: getfirst
```

Более того, эти процедуры позволяют, соответственно, инициировать методы `getFirst`, `setFirst`, `getSecond` и `setSecond` любого объекта, который их реализует.

Можно упростить общие процедуры вызова методов объектов, реализующих связанную пару значений, использовав возможность Maple определять новые типы. В нашем случае определим тип `pair` как тип модуля, экспортирующего четыре процедуры — `getFirst`, `setFirst`, `getSecond` и `setSecond`:

```
> `type/pair` := '`module`( getFirst, getSecond, setFirst, setSecond)':
```

Тогда наши общие процедуры перепишутся в следующем виде, в котором проверка существования соответствующих процедур в передаваемых объектах будет возложена на Maple:

```
> first := proc( obj::pair)
            obj:-getFirst()
      end proc:
      second := proc( obj::pair)
            obj:-getSecond()
      end proc:
      setfirst := proc( obj::pair, v)
            obj:-setFirst(v)
      end proc:
      setsecond := proc( obj::pair, v)
            obj:-setSecond(v)
      end proc:
> first(1);
Error, (in first) module does not export `getFirst`
```

Внимание!

В справочной системе Maple есть файлы с примерами создания объектов и пакетов. Посмотрите файлы справки `?examples/obj`, `?examples/memo` и `?examples/binarytree`.

5.4.2. Вызов внешних процедур

Одной из привлекательных возможностей, включенных в версию Maple 6, является возможность вызова внешних откомпилированных и сохраненных в статической библиотеке процедур, написанных на языке С. Организация подобного взаимодействия осуществляется с помощью новой технологии Maple, состоящей в том, что с помощью специальной команды `define_external()` генерируется пара интерфейсных функций — одна написанная на языке Maple, а вторая на С. Они совместно реализуют механизм вызова внешней функции: передачу ей значений в требуемом формате языка С и возврат значений в формате Maple. При работе в операционной системе Windows сгенерированная Maple на языке С интерфейсная функция компилируется и вместе с внешней процедурой С, которую пользователь желает вызвать из рабочего листа Maple, помещается в динамически подключаемую библиотеку DLL. Эта библиотека при вызове внешней процедуры С подключается к выполняющемуся ядру Maple и находящаяся в ней интерфейсная функция выполняет вызов процедуры С.

Алгоритм вызова внешней функции таков. Если пользователь вызывает внешнюю процедуру С, то автоматически осуществляется вызов интерфейсной функции, написанной на Maple, которая инициирует интерфейсную функцию, написанную на С. Эта последняя осуществляет преобразование типов передаваемых в вызываемую процедуру данных в формат языка С и затем вызывает внешнюю процедуру через встроенную функцию `call_external`.

После завершения выполнения внешней процедуры интерфейсная С-функция преобразует возвращаемые данные в формат Maple и передает управление интерфейсной Maple-функции, которая и осуществляет непосредственный возврат полученных значений на рабочий лист Maple.

Замечание

Именно с использованием подобного механизма пакет `LinearAlgebra` обращается к составляющим его программам численных методов линейной алгебры пакета NAG.

Команда `define_external()`, конструирующая и возвращающая функцию, которую в дальнейшем пользователь может использовать для действительного обращения к внешней процедуре С, имеет следующий синтаксис:

```
define_external(имя_функции, apr1, ..., aprN, тип_возвр_значения, опции)
```

Параметр `имя_функции` задает имя вызываемой внешней процедуры. Следующие за ним параметры вида `aprN`, где `N` — целое число, описывают формальные параметры вызываемой процедуры в том порядке, в каком они определены во внешней процедуре. Возвращаемое значение функции описывается параметром `тип_возвр_значения` аналогично описанию типов формальных параметров. Последними задаются опции в форме уравнений, которые уточняют некоторые детали: имя библиотеки, содержащей внешнюю процедуру, расположение компилятора С и его некоторые опции и т. п.

Внимание!

Для вызова внешних процедур необходим установленный на компьютере пользователя компилятор С. Maple распространяется с заранее предопределенной командой вызова компилятора С для компиляции и сборки интерфейсной С-функции. Для разных операционных систем предусматриваются разные компиляторы, например, для операционной системы Windows предполагается компилятор С фирмы Microsoft. Опции команды `define_external()` позволяют использовать компилятор, отличный от используемого по умолчанию.

Каждый формальный параметр внешней процедуры описывается в виде:

```
идентификатор::дескриптор
```

Идентификатор представляет собой имя переменной величины Maple, которой не присвоено никакого значения. Для возвращаемого значения внешней процедуры используется идентификатор `RETURN`.

Дескриптор описывает тип данных формального параметра, простой или сложный. К сложным типам данных в С относятся массив, структура, объединение и перечисляемый тип.

Дескриптор простого типа данных представляет имя типа с указанием в квадратных скобках отведенного количества байтов для хранения этого типа во внешней процедуре:

- `boolean[кол_байт]` — булевский тип, принимающий значение `true` (1) или `false` (0).
- `integer[кол_байт]` — целое со знаком, количество отводимых байтов зависит от используемого компилятора С и обычно принимает значение 2, 4 или 8. Соответствующие значения Maple должны быть типа `integer`.
- `float[кол_байт]` — вещественное число с плавающей точкой, количество отводимых байтов зависит от используемого компилятора С и обычно принимает значение 4, 8, 10, 12 или 16. Соответствующие значения Maple должны быть типа `numeric`.
- `complex[кол_байт]` — пара вещественных чисел с плавающей точкой с заданным количеством отведенных для хранения байтов. Соответствующие значения Maple должны быть типа `complex`.
- `char[кол_байт]` — один символ, для хранения которого отводится заданное число байтов. Если параметр `кол_байт` имеет значение большее, чем `1`, то старшие байты должны содержать нули, тогда как в младшем байте должен содержаться код символа. Соответствующие значения Maple должны быть типа `string`, содержащие только один символ.
- `string[кол_байт]` — строка символов, которая в С представляется массивом символов, но с ней можно манипулировать как единым целым. Параметр `кол_байт` определяет длину строки (количество содержащихся в ней символов), и если задан, то соответствует строке С фиксированной длины. Если он опущен, то внешняя процедура ожидает получения строки переменной длины. Соответствующие значения Maple должны быть типа `string` или `symbol`.

Для описания массива — сложного типа, предназначенного для хранения однотипных простых данных и предоставляющего доступ к ним с помощью индексов, — применяется дескриптор следующего вида:

```
ARRAY(разм1, ..., размN, дескриптор_данных, опции)
```

Каждый параметр `размN`, где `N` — целое число, задается в виде целого диапазона и определяет изменение индекса соответствующего размерения. Нижняя или верхняя граница любого диапазона может быть именем любого формального параметра вызываемой внешней процедуры. В этом случае границы изменения индекса будут установлены во время выполнения процедуры.

Параметр `дескриптор_данных` описывает тип данных, содержащихся в массиве, и может быть любым типом, используемым в конструкторах `Array`, `Matrix` или `Vector` соответствующих типов Maple: `integer[n]`, `float[n]` и `complex[n]`. Здесь `n` является целым числом, определяющим требуемое для хранения каждого элемента массива количество байтов оперативной памяти.

Опции задают ряд соглашений относительно массива и соответствуют допустимым опциям конструктора `Array()` в Maple с единственным исключе-

нием для опции, определяющей индексную функцию, которая инициализирует элементы массива. Ее следует задавать в виде indfn. Кроме стандартных опций конструктора Array() можно использовать две дополнительные опции — COPY и NO_COPY, которые определяют, будет ли передаваться во внешнюю процедуру копия массива или не будет.

Структура представляет коллекцию не однотипных данных. Ее дескриптор выглядит следующим образом:

```
STRUCT(поле1::дескриптор_данных1, ..., полеN::дескриптор_данныхN)
```

Каждая пара поле::дескриптор_данных описывает соответствующее поле структуры.

Объединение похоже на структуру за исключением того, что все поля используют одну и ту же область памяти. Параметры дескриптора объединения аналогичны параметрам дескриптора структуры:

```
UNION(поле1::дескриптор_данных1, ..., полеN::дескриптор_данныхN)
```

Таблица (table) и список (list) Maple могут использоваться в качестве фактических параметров, если формальные параметры объявлены как структуры. Для объединений допустимым типом Maple является только тип table.

Так как Maple не поддерживает перечисляемый тип, то следует использовать тип integer[n], в котором количество байт n должно соответствовать размеру целого типа, используемому для реализации перечисляемого типа компилятора, с помощью которого была откомпилирована внешняя процедура.

По умолчанию все параметры во внешнюю процедуру передаются по значению. Если через какой-либо параметр возвращается некоторое вычисленное в процедуре значение, то он должен быть объявлен как передаваемый по ссылке с помощью модификатора REF:

```
идентификатор::REF(дескриптор, опции)
```

Допустимыми опциями могут быть ANYTHING, CALL_ONLY или RETURN_ONLY. Первая опция эквивалентна параметру С-процедуры, определенному как void *. Вторая опция хотя и определяет передачу параметра по ссылке, однако запрещает возврат через него вычисленных во внешней процедуре величин, предохраняя тем самым фактический параметр от изменения. Последняя опция RETURN_ONLY предопределяет, что в действительности через этот параметр никаких данных в процедуру не передается, он служит только для возврата значений из внешней процедуры.

Опция LIB команды define_external() задает в виде строки Maple полный путь и имя библиотеки, в которой размещена внешняя вызываемая процедура. Кроме нее можно использовать опцию COMPILER, определяющую компилятор, с помощью которого будет компилироваться интерфейсная С-функция, а также опции, изменяющие установленные по умолчанию ключи компилятора при вызове его из командной строки. Опция OPTIONS служит для зада-

ния любых ключей компилятора за исключением -I и -Fe, для которых предназначены, соответственно, опции INC_FLAG и OBJ_FLAG.

В качестве примера обращения к внешней процедуре откомпилируем и занесем в библиотеку mat_mult.lib следующую процедуру перемножения двух квадратных матриц, написанную на языке С и возвращающую результирующую матрицу произведения через свой параметр:

```
void mat_mult(double *A, double *B, double *C, int I, int J, int K)
{ int i,j,k;
  double t;
  for(i=0;i<I;++i)
     for(k=0;k<K;++k) {
       t = 0.0;
       for(j=0;j<J;++j)
          {t += A[i*J+j]*B[j*K+k];}
       C[i*K+k] = t;}
}
```

Вызов команды define_external() создания интерфейсных функций выглядит следующим образом:

```
> mat_mult:=define_external(`mat_mult`,
       a::ARRAY(1..i,1..j,float[8]),
       b::ARRAY(1..j,1..k,float[8]),
       c::REF(ARRAY(1..i,1..j,float[8]),RETURN_ONLY),
       i::integer[4],
       j::integer[4],
       k::integer[4],
       LIB="D:\\maple6math&mech\\mat_mult.lib");
  mat_mult := proc(p1::{rtable, name, identical(0)},
       p2::{rtable, name, identical(0)}, p3::{name, identical(0)},
       p4::integer, p5::integer, p6::integer)
       option call_external;
       call_external(1073812720, 1073799584, false, args)
       end proc
```

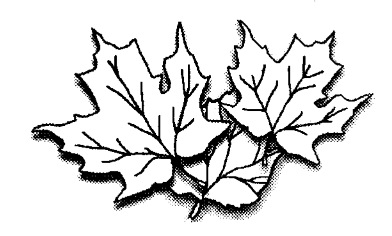
Теперь можно создать две матрицы с помощью конструктора Matrix(), в котором необходимо указать тип данных float[8] и опцию хранения матриц по строкам, как принято в С, а не по столбцам, как принято в Maple. Обращение к созданной процедуре mat_mult() осуществит перемножение матриц с помощью внешней процедуры mat_mult(), написанной на языке С:

```
> A := Matrix(100,100,(i,j)->i/j, datatype=float[8], order=C_order):
> B := Matrix(100,100,(i,j)->i*j, datatype=float[8], order=C_order):
> mat_mult(A,B,C,100,100,100):
```

Результат перемножения двух матриц `A` и `B` будет помещен в автоматически созданную процедурой `mat_mult()` матрицу `C`.

Замечание

Приведенные здесь результаты создания интерфейсных функций командой `define_external()` взяты из Справки Maple. Автору, к сожалению, не удалось самостоятельно выполнить вызов внешней процедуры из-за непонятных проблем, связанных с компилятором Microsoft Visual C++ 6.0. Команда `define_external()` вызывала компилятор С, транслировала сгенерированную Maple интерфейсную С-функцию, создавала библиотеку DLL и помещала в нее оттранслированную функцию и внешнюю процедуру Maple перемножения матриц, но на этапе подключения ее к ядру Maple возникала непонятная ошибка, которую автору не удалось исправить.



ГЛАВА 6

Maple в Excel

В настоящее время фирмы, разрабатывающие прикладные приложения, стремятся разными способами интегрировать свои продукты с приложениями других фирм для расширения функциональных возможностей получаемого совокупного "продукта". Это достигается либо с использованием технологии Automation при разработке собственного продукта, либо разработкой дополнительных модулей (add-in, plug-in), включаемых в приложения сторонних фирм. Примерами подобного сотрудничества являются: Maple и Matlab, Maple и Mathcad, Maple и Excel и др.

Электронные таблицы MS Excel являются одним из популярных офисных приложений для работы с табличными данными. В них можно осуществлять не только простейшие арифметические операции с группами ячеек данных в таблицах, но и производить достаточно сложные вычисления и анализ данных с помощью так называемых надстроек (add-in) — специальным образом разработанных документов Excel, хранимых в файлах с расширением xla. Читателю, работавшему с MS Excel, вероятно, хорошо знакомы такие надстройки, как **Поиск решения** и **Анализ данных**. (Относительно первой из них следует заметить, что реализованный в ней универсальный алгоритм поиска оптимального решения нелинейных задач не всегда корректно работает, когда применяется к задачам линейного программирования.) Добавление новой функциональности в приложение Excel с помощью надстроек позволяет сторонним фирмам предоставить возможности своих собственных приложений пользователям программы Excel, не покидая его среду. Именно таким путем пошла фирма Maplesoft, включив в новую версию своего продукта Maple 6 надстройку для приложения MS Excel 2000 из пакета MS Office 2000.

Эта надстройка позволяет выполнять любую команду любого пакета Maple непосредственно из рабочего листа Excel и получать результаты ее выполнения в ячейках рабочего листа для дальнейшего использования, строить графики любых функций простым заданием в ячейке ее математического вы-

ражения, а не формируя таблицу значений, необходимую для построения графика функции с помощью диаграмм Excel, переносить данные рабочего листа Excel в приложение Maple либо в виде объекта-матрицы, либо помещать их непосредственно в электронную таблицу Maple и наоборот. Подобное взаимодействие двух популярных приложений существенно расширяет функциональность одного (Excel) и расширяет область использования другого (Maple).

6.1. Установка и получение справки

Установить надстройку Maple 6 для Excel 2000 достаточно просто. Прежде всего необходимо, чтобы на компьютере пользователя была установлена программа Excel 2000 из семейства приложений Office 2000. Если это так, то обычная процедура установки Maple 6 сделает все необходимые действия, чтобы надстройка Maple 6 была доступна из Excel 2000.

После установки программы Maple 6 в диалоговом окне **Надстройки** приложения Excel (команда **Сервис ➤ Надстройки**) в списке доступных надстроек появится запись **Maple 6 Excel Add-in**. Для ее подключения следует установить расположенный слева от нее флажок и нажать кнопку **ОК**. В результате выполнения указанных действий будет загружена надстройка Maple 6, и отобразится панель инструментов для работы с Maple, внешний вид которой показан на рис. 6.1.



Рис. 6.1. Панель инструментов **Maple 6**

Замечание

Если по каким-то причинам в списке доступных надстроек не окажется надстройки Maple, то следует нажать кнопку **Обзор** диалогового окна **Надстройки** и выбрать файл wmimplex.XLA, расположенный в подкаталоге Excel основного каталога, в котором установлена программа Maple 6.

Замечание

Отобразить или скрыть панель инструментов **Maple 6** можно командой Excel **Сервис ➤ Настройка**. На вкладке **Панели инструментов** отображаемого ею диалогового окна **Настройка** следует установить или сбросить флажок рядом с панелью инструментов **Maple 6** в списке всех доступных панелей инструментов.

Внимание!

Надстройка Maple 6 работает только в MS Excel 2000. В предыдущей версии Excel 97 она будет доступна в диалоговом окне **Надстройки**, но попытка ее активизировать окажется неуспешной.

Кнопка ![icon] панели инструментов предназначена для работы со справкой Maple. Ассоциированная с ней команда отображает диалоговое окно **Maple 6 Help** (рис. 6.2), в котором можно получить справку по любой доступной команде Maple, а также общую справку по работе с надстройкой Maple 6 нажатием кнопки **Using Maple in Excel**.

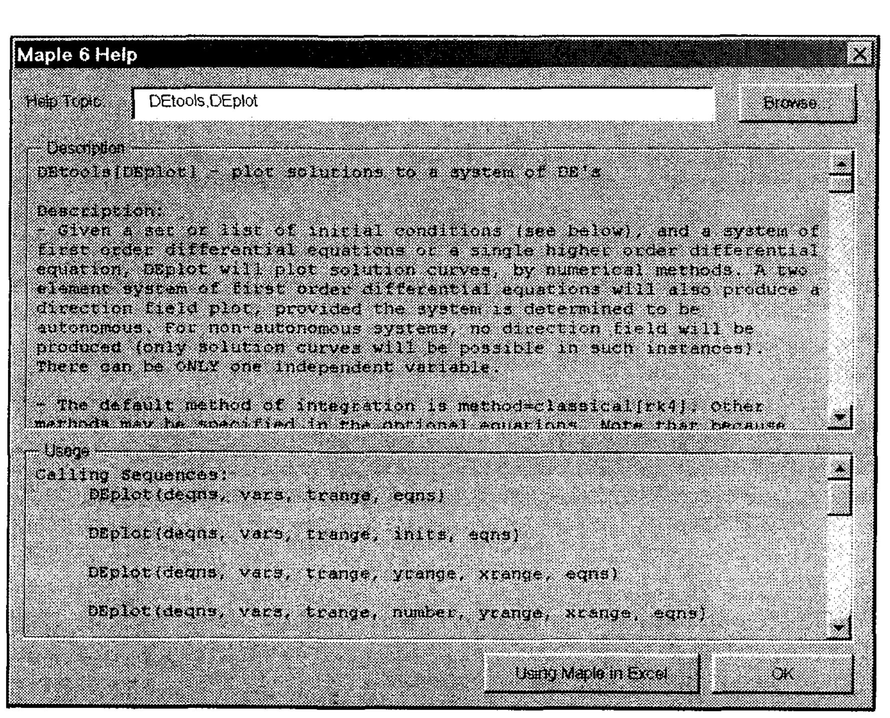


Рис. 6.2. Диалоговое окно справки Maple в Excel

Общая справка отображается в стандартном диалоговом окне справки операционной системы Windows и имеет четыре раздела, содержащих общую информацию об использовании Maple в Excel (**General Information**), описание вызова функций Maple из рабочих листов Excel и задание их параметров ссылками на содержимое ячеек рабочего листа Excel (**Using Maple 6**), необходимую информацию по настройке параметров ядра Maple (**Customizing Maple 6**) и соответствие типов данных Maple и Excel (**References**).

Для получения справки по конкретной команде Maple в диалоговом окне **Maple 6 Help** в поле **Help Topic** следует набрать имя команды и нажать клавишу <Enter>. В расположенных ниже поля ввода двух областях **Description** и **Usage** отобразится описание команды и синтаксис ее вызова. Например, на рис. 6.2 можно видеть информацию по команде отображения решения обыкновенного дифференциального уравнения `DEplot()` из пакета `DEtools`.

Замечание

Имя команды следует задавать точно в соответствии с ее названием в Maple, используя там, где необходимо, прописные буквы. В противном случае можно не получить справку по интересующей команде.

Замечание

При задании имени команды можно перед ней через запятую указывать имя пакета Maple, в котором она расположена.

Если пользователь забыл имя команды, или не знает, какие команды в Maple предназначены для выполнения необходимых действий, можно воспользоваться окном отображения доступных команд **Maple 6 Help Browser**, нажав кнопку **Browse**. Это окно (рис. 6.3) полностью соответствует окну поиска информации в справке самого приложения Maple: при выборе темы в последнем заполненном списке в следующем по порядку списке отображаются подчиненные темы, если они существуют, или становится доступной кнопка **Display Topic**, нажатие на которую приводит к отображению справки по выбранной команде.

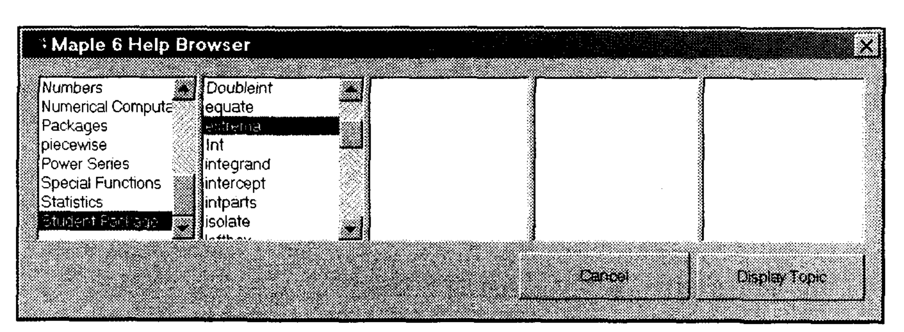


Рис. 6.3. Окно отображения доступных команд **Maple 6 Help Browser**

Кроме описания команд и их синтаксиса в Справке Maple можно получить общую информацию о том, как работать с системой Maple в Excel, краткую информацию о доступных пакетах и их назначении и многое другое.

6.2. Функции Maple на рабочем листе Excel

Для вычисления выражений Maple, которые могут представлять любые синтаксически допустимые конструкции и вызовы команд Maple, их следует задавать в ячейках рабочего листа Excel, используя функцию рабочего листа `Maple()`, доступную после подключения надстройки Maple 6. В простейшем случае в качестве параметра этой функции задается один оператор Maple, заключенный в двойные кавычки. Результат его вычисления или сообщение об ошибке будет помещено в ту же самую ячейку, в которой была задана

функция `Maple()`. Например, если необходимо вычислить производную функции $\sin(x)^2$, то достаточно в любой ячейке задать обращение к следующей функции `Maple()`:

```
=Maple("diff(sin(x)^2,x))
```

Здесь знак равенства (=) перед именем вызываемой функции требуется по правилам обращения к функциям рабочего листа Excel. Передаваемый параметр — это заключенный в двойные кавычки оператор Maple обращения к команде дифференцирования `diff()`. Обратите внимание, что его можно не завершать разделителем (;), требуемым по правилам синтаксиса Maple. После ввода формулы в ячейку она автоматически будет вычислена, как только мы покинем содержащую ее ячейку, щелкнув указателем мыши на другой ячейке рабочего листа или нажав любую из клавиш со стрелками или <Enter>. Результат вычисления отобразится в ячейке с введенной формулой (рис. 6.4).

A2	=Maple("diff(sin(x)^2,x)")	
	A	B	C
1			
2	2*sin(x)*cos(x)		
3			

Рис. 6.4. Ячейка с выражением Maple и результатом его вычисления

При обращении к функции Maple можно задать несколько операторов, разделенных двоеточиями или точками с запятыми. Как читатель помнит, при работе в Maple, если оператор завершается двоеточием, то результаты его вычисления не отображаются в области вывода, а завершение оператора точкой с запятой, наоборот, отображает в ней результаты его выполнения, причем результат отображения каждого оператора осуществляется в новой строке области вывода. В Excel действуют эти же правила, за исключением того, что отображение результатов выполнения цепочки операторов осуществляется в одну строку, в которой символ □ представляет символ перехода на новую строку (рис. 6.5).

A1	=Maple("y:=sin(x)^2,x; z:=diff(y,x);int(z,x)")		
	A	B	C	D
1	y := sin(x)^2, x□z := 2*cos(x)^2-2*sin(x)^2□2*sin(x)*cos(x)			
2				

Рис. 6.5. Отображение результатов выполнения нескольких операторов в одну строку

Для того чтобы результаты выполнения операторов цепочки отображались в отдельных строках, следует на вкладке **Выравнивание** диалогового окна **Формат ячеек**, отображаемого командой **Формат ячеек**, установить флажок **переносить по словам**. При этом все символы □ в строке содержимого ячейки будут интерпретированы как символы перехода на новую строку (рис. 6.6).

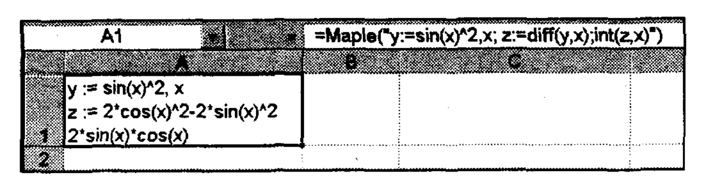


Рис. 6.6. Отображение результатов выполнения каждого оператора в отдельной строке

Результаты выполнения операторов Maple отображаются в ячейке рабочего листа по умолчанию в текстовом формате, даже если результатом их выполнения является число. Это последнее утверждение важно для английской, французской и немецкой локализованных версий Excel, в которых в качестве разделителя целой и дробной части вещественного числа используется символ точка. В русской версии для этих целей используется запятая, поэтому, естественно, что вещественное число с десятичной точкой, полученное в результате выполнения оператора Maple, представлено на листе в текстовом формате. Этот числовой результат нельзя, к сожалению, использовать для дальнейших вычислений в функциях Excel. Можно настроить ядро Maple таким образом, чтобы результат вычислений представлялся в числовом формате Excel. Однако при этом может быть потеряна точность, так как числа формата Excel представляются с меньшим числом значащих цифр в мантиссе, чем в Maple. Использование числового формата Maple имеет одно неоспоримое преимущество: в этом режиме вычислений можно установить произвольное количество значащих цифр в мантиссе числа. Опять-таки, все вышесказанное не затрагивает русскую локализованную версию. Она не реагирует на переключение между числовым форматом Maple и Excel в связи с использованием запятой для отделения дробной части вещественных чисел, хотя позволяет производить вычисления с произвольным количеством значащих цифр в мантиссе вещественных чисел. (О настройке параметров Maple в Excel *см. раздел 6.3.*)

На рис. 6.7 показан фрагмент рабочего листа Excel с результатами вычислений операторов Maple, заданных в ячейках столбца `A` (в соответствующих ячейках столбца `B` приведены обращения к функции `Maple()`, которые задавались для выполнения вычислений). Обратите внимание, что Maple по умолчанию возвращает результаты с использованием точной арифметики (ячейка `A2`). Для получения результата в форме действительных чисел следует, как всегда в Maple, использовать функцию `evalf()` (ячейки `A4` и `A6`). Вычисление оператора Maple в ячейке `A4` осуществлялось в режиме с установленной длиной мантиссы 30 цифр, тогда как значение в ячейке `A6` получено при длине мантиссы 10. При вычислении выражений Maple в ячейках рабочего листа можно так же, как и непосредственно в самом Maple, использовать различные встроенные константы, например, число π (см. ячейку `A8`).

Основные преимущества в использовании электронных таблиц заключаются в том, что можно использовать значения одних ячеек при вычислении формул в других ячейках. Первые ячейки называются влияющими, вторые —

зависимыми. Этот же механизм влияющих и зависимых ячеек можно использовать и при вычислении команд Maple на рабочем листе Excel. Синтаксис вызова функции `Maple()` с использованием значений, содержащихся в других ячейках рабочего листа, имеет следующий вид:

```
=Maple("выражение_Maple" <разделитель><ссылка><разделитель><ссылка> ...)
```

Здесь <разделитель> — это символ, используемый в качестве разделителя элементов в списке Excel. В английской версии таковым является запятая (,), в русской локализованной версии для этих целей используется точка с запятой (;).

A8 =Maple("sin(Pi/2)")

	A	B	C
1			
2	exp(1)	=Maple("exp(1)")	Абсолютная точность
3			
4	2.71828182845904523536028747135	=Maple("evalf(exp(1))")	Точность Maple 30 знаков
5			
6	2.718281828	=Maple("evalf(exp(1))")	Точность Maple 10 знаков
7			
8	1	=Maple("sin(Pi/2)")	Использование констант
9			

Рис. 6.7. Вычисления с различной точностью

Параметр <ссылка> представляет относительную или абсолютную ссылку на ячейку рабочего листа. Напомним, что в Excel каждая ячейка имеет собственный адрес, состоящий из наименования столбца и номера ряда, на пересечении которых расположена ячейка, например, адресом ячейки, расположенной на пересечении столбца `A` и ряда `4`, является `A4`. Этот адрес и используется для организации ссылок на ячейки. Различают абсолютные и относительные ссылки. Абсолютная ссылка задает место ячейки на рабочем листе абсолютно, т. е. независимо от местоположения на рабочем листе ячейки, из которой осуществляется ссылка на нее. Для этого перед наименованием столбца и номером строки адреса ячейки ставятся знаки доллара (`$`), например, `$A$1` определяет абсолютную ссылку на ячейку `A1`. Если в качестве ссылки используется адрес ячейки без знаков доллара, то создается относительная ссылка, которая определяет адрес влияющей ячейки относительно зависимой. Разница между абсолютной и относительной ссылкой прежде всего проявляется при копировании содержимого зависимой ячейки (под копированием подразумевается, в том числе, и автозаполнение соседних ячеек путем перетаскивания маркера заполнения). Абсолютная ссылка всегда указывает на одну и ту же ячейку, тогда как относительная ссылка вычисляет адрес ячейки, расположенной аналогично по отношению к ячейке, в которую копируется содержимое. Например, если скопировать ячейку `C3`, содержащую относительную ссылку на `A1`, в ячейку `D6`, то `D6` будет содержать ссылку на `B4`. Соответственно, если поставить знак доллара только

перед буквой или цифрой в ссылке на ячейку, то при копировании станется неизменной только эта часть адреса.

Ссылка в функции `Maple()` может представлять ссылку на одну ячейку того же рабочего листа, на котором вычисляется выражение Maple (например, `B2`, `$A$2`), на диапазон ячеек (например, `A2:B4`) или даже на ячейку или диапазон другого листа (например, `Лист2!A2:B4`).

В выражении Maple, задаваемом первым параметром, сослаться на содержимое ячеек, перечисленных в обращении к функции `Maple()`, можно с помощью конструкции `&n`, где `n` — натуральное число, представляющее порядковый номер ссылки на ячейку. Например, если в ячейке `A2` содержится строковое значение `sin(x)`, а в ячейке `B2` — `x`, то формула `=Maple("diff(&1,&2); A2; B2)` вычислит производную функцию `sin(x)` по переменной `x`.

Применение относительных и абсолютных ссылок в выражениях Maple позволяет использовать преимущества такого средства Excel, как протягивание. Например, предположим, что нам необходимо быстро создать таблицу значений функции и ее производной на заданном интервале изменения независимой переменной с заданным шагом. Для решения данной задачи занесем в ячейку `B3` выражение, задающее функцию, например, `sin(x)`. В ячейке `A3` будем хранить имя независимой переменной (в нашем случае `x`). В диапазоне `A4:A14` зададим с шагом `0.1` значения независимой переменной от `0` до `1` (в русской версии рекомендуем установить текстовый формат всех ячеек диапазона). В ячейке `B4` зададим вычисление значения функции из ячейки `B3`:

```
=Maple("evalf(eval(&1,&2=&3)); $B$3; $A$3; A4)
```

В ячейке `C4` будем вычислять производную функции из ячейки `B3` по переменной из ячейки `A3`:

```
=Maple("evalf(eval(diff(&1,&2), &2=&3)); $B$3; $A$3; A4)
```

Результат наших действий представлен на рис. 6.8.

C4 =Maple("evalf(eval(diff(&1,&2), &2=&3))"; B3; A3;A4)

	A	B	C	D	E	F
1						
2	Аргумент	Функция	Производная			
3	x	sin(x)	cos(x)			
4	0	0.	1.			
5	0.1					

Рис. 6.8. Вычисление с использованием ссылок на ячейки

Теперь достаточно выделить ячейку `B4` и протянуть маркер заполнения до ячейки `B14`. Так как ссылки на ячейку, содержащую функцию, и ячейку с независимой переменной абсолютные, то в выделенные ячейки скопируют-

ся ссылки именно на эти ячейки. Ссылка на значение независимой переменной является относительной, поэтому в каждой последующей ячейке будет сформирована ссылка на соответствующее значение независимой переменной из первого столбца. Результатом протягивания будет столбец значений функции. Аналогичную процедуру произведем и с ячейкой C4. Получим столбец значений производной. Результат можно увидеть на рис. 6.9.

C14 =Maple("evalf(eval(diff(&1,&2), &2=&3))"; B3; A3;A14)

	A	B	C	D	E	F
1						
2	Аргумент	Функция	Производная			
3	x	sin(x)	cos(x)			
4	0	0.	1.			
5	0.1	.9983341665e-1	.9950041653			
6	0.2	.1986693308	.9800665778			
7	0.3	.2955202067	.9553364891			
8	0.4	.3894183423	.9210609940			
9	0.5	.4794255386	.8775825619			
10	0.6	.5646424734	.8253356149			
11	0.7	.6442176872	.7648421873			
12	0.8	.7173560909	.6967067093			
13	0.9	.7833269096	.6216099683			
14	1.0	.8414709848	.5403023059			
15						

Рис. 6.9. Таблица значений функции и ее производной

Обратите внимание на формулу, хранящуюся в ячейке C14 и отображаемую в строке формул. Последней ссылкой является ссылка на ячейку A14, которая была сформирована в результате протягивания ячейки C4 до ячейки C14.

Теперь можно обратиться к мастеру диаграмм и построить графики функции и ее производной, а можно воспользоваться функцией plot() пакета Maple и сразу же построить графики функции и ее производной:

```
=Maple("plot([&1,diff(&1,&2)],&2=&3..&4); B3; A3; A4; A14)
```

На рис. 6.10 показан график, построенный системой Maple на листе Excel. Его можно перетаскивать с одного места на другое, так как по умолчанию он строится на графическом слое рабочего листа, а также можно изменять его размеры, используя маркеры размеров, отображаемые при выделении графика на рабочем листе щелчком мыши.

Использование в вызовах команд Maple ссылок на ячейки Excel позволяет обращаться к еще одной замечательной способности Excel: автоматический пересчет формул при изменении значения влияющей ячейки. Так, если в ячейке задать другую функцию, то автоматически на рабочем листе осуществится пересчет таблицы и будет построен график новой функции и ее производной.

Чтобы предоставить читателю возможность сравнить графические возможности Maple и Excel, мы приводим на рис. 6.11 графики функции $y=x/(x^2+1)$ на интервале [-1,1].

Microsoft Excel - Таблица

Файл Правка Вид Вставка Формат Сервис Данные Окно Справка

A1 =Maple("plot([&1,diff(&1,&2)],&2=&3..&4)"; B3; A3; A4; A14)

	A	B	C
1	Maple Plot		
2	Аргумент	Функция	Производная
3	x	sin(x)	cos(x)
4	0	0.	1.
5	0.1	.9983341665e-1	.9950041653
6	0.2	.1986693308	.9800665778
7	0.3	.2955202067	.9553364891
8	0.4	.3894183423	.9210609940
9	0.5	.4794255386	.8775825619
10	0.6	.5646424734	.8253356149
11	0.7	.6442176872	.7648421873
12	0.8	.7173560909	.6967067093
13	0.9	.7833269096	.6216099683
14	1.0	.8414709848	.5403023059

Лист1 Лист2 Лист3

Готово

Рис. 6.10. График, построенный с помощью Maple

Microsoft Excel - Диаграммы

Файл Правка Вид Вставка Формат Сервис Данные Окно Справка

A1 =Maple("plot(x/(x^2+1),x=-1..1,numpoints=70)")

A1: Maple Plot

	D	E
1	-1	-0,5
2	-0,9	-0,49724
3	-0,8	-0,4878
4	-0,7	-0,4698
5	-0,6	-0,44118
6	-0,5	-0,4
7	-0,4	-0,34483
8	-0,3	-0,27523
9	-0,2	-0,19231
10	-0,1	-0,09901
11	0	0
12	0,1	0,09901
13	0,2	0,192308
14	0,3	0,275229
15	0,4	0,344828
16	0,5	0,4
17	0,6	0,441176
18	0,7	0,469799
19	0,8	0,487805
20	0,9	0,497238
21	1	0,5

График, построенный Maple

Диаграмма, построенная Excel

Лист1 Лист2 Лист3

Готово

Рис. 6.11. Графики, построенные с помощью Maple и Excel

Обратите внимание на то, что для построения диаграммы Excel пришлось сначала создать таблицу значений функции, а потом с помощью мастера диаграмм построить график функции и произвести далее его небольшую корректировку. На что, с нашей точки зрения, ушло больше времени, чем на ввод обращения к функции `plot()` в ячейке `A2`.

Кроме непосредственного ввода обращения к функции `Maple()`, реализующей вызов команд системы Maple, надстройка Excel для работы с ней предоставляет возможность использовать мастер функций Maple. Его вызов осуществляется нажатием кнопки на панели инструментов. При этом отображается диалоговое окно доступных команд **Maple 6 Function Browser**, которое идентично окну **Maple 6 Help Browser** при работе со справочной системой Maple в Excel. Единственное отличие заключается в том, что в нем отображается подмножество доступных команд из трех разделов: **Graphics**, **Mathematics** и **Programming** (рис. 6.12).

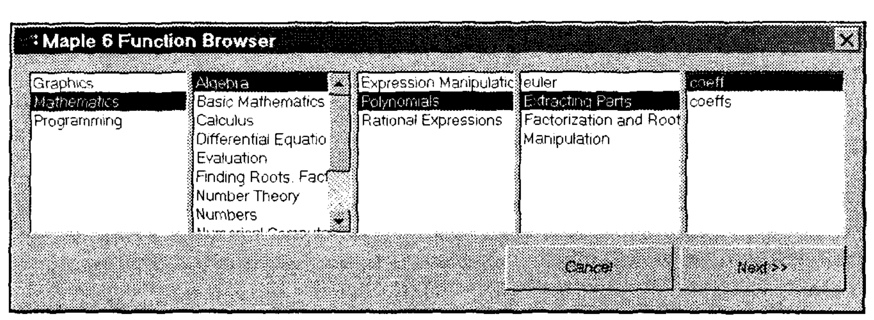


Рис. 6.12. Окно доступных команд

После выбора необходимой команды и нажатия кнопки **Next** отображается следующее окно мастера функций Maple. В раскрывающемся списке **Calling Sequence** следует выбрать форму вызова функции и задать требуемые параметры в полях **Required Parameters**, которые представляют собой специальные ссылочные поля Excel, позволяющие быстро вставить ссылку на требуемую ячейку. Если нажать на кнопку в правой части поля, то все диалоговое окно свернется до величины этого поля, чтобы открыть больший обзор рабочего листа, на котором следует щелчком кнопки мыши выбрать нужную ячейку с требуемым значением параметра. После этого необходимо снова нажать кнопку в правой части поля, чтобы окно приняло первоначальный вид. Значение параметра можно вводить в поля и непосредственно, без ссылок на ячейки, установив курсор в требуемое поле (рис. 6.13).

Нажатие кнопки **OK** приведет к немедленному вычислению введенной команды с заданными параметрами. Если какой-либо параметр пропущен, то отобразится диалоговое окно с подсказкой ввести его значение.

На этом же диалоговом окне отображается краткая справка о назначении команды и описание всех ее параметров.

Maple Function Wizard: coeff

Calling Sequence: coeff(p, x, n)

Function Description
the coefficients of a polynomial

Parameter Descriptions
p: a polynomial in x
x: an algebraic expression
n: an integer representing the power of x

Required Parameters:
p: Лист1!A16
x: x
n: 1

OK Cancel

Рис. 6.13. Диалоговое окно второго шага мастера функций Maple

Кроме выполнения команд Maple надстройка позволяет экспортировать данные, содержащиеся в ячейках рабочего листа в Maple, и наоборот, вставлять данные из электронной таблицы Maple в рабочий лист Excel.

Экспорт данных осуществляется нажатием кнопки на панели инструментов Maple (первая кнопка со всплывающей подсказкой **Copy From Excel To Maple**) при выделенном диапазоне ячеек на рабочем листе. Вставка экспортированных данных в рабочий лист Maple осуществляется командой **Edit ➤ Paste** либо в область ввода, либо в электронную таблицу (рис. 6.14).

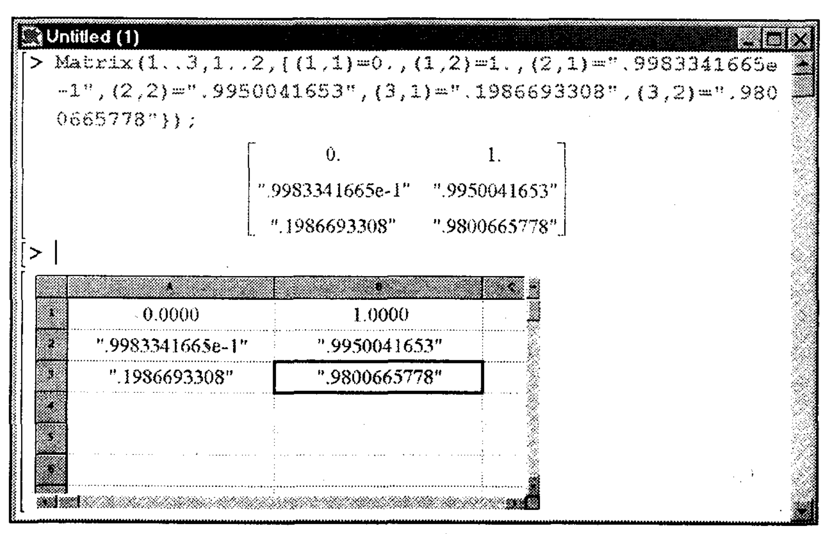


Рис. 6.14. Вставленные в рабочий лист Maple данные из рабочего листа Excel

При вставке данных из Excel в область ввода рабочего листа Maple они преобразуются в объект `Matrix()`, причем при использовании русской версии Excel все данные, даже являющиеся числами, представляются в форме текстовых строк.

Импортировать данные из Maple в рабочий лист Excel можно с помощью кнопки панели инструментов Maple (вторая кнопка с всплывающей подсказкой **Paste From Maple Spreadsheet To Excel**), предварительно скопировав диапазон данных из электронной таблицы Maple в Буфер обмена командой **Edit** ➢ **Copy**. При этом даже в русской версии данные будут представлены правильными числами, т. е. на листе Excel десятичная точка Maple будет преобразована в десятичную запятую.

6.3. Настройка параметров Maple в Excel

Нам осталось рассмотреть назначение последней кнопки на панели инструментов Maple — кнопки . Он предназначена для отображения диалогового окна **Maple 6 Excel Options**, на вкладках которого можно установить разные параметры и опции Maple для *всего* сеанса работы в Excel.

Первая вкладка **Packages** этого окна (рис. 6.15) предназначена для подключения пакетов Maple и полностью соответствует команде `with()`, выполняемой на рабочем листе Maple для подключения требуемого пакета.

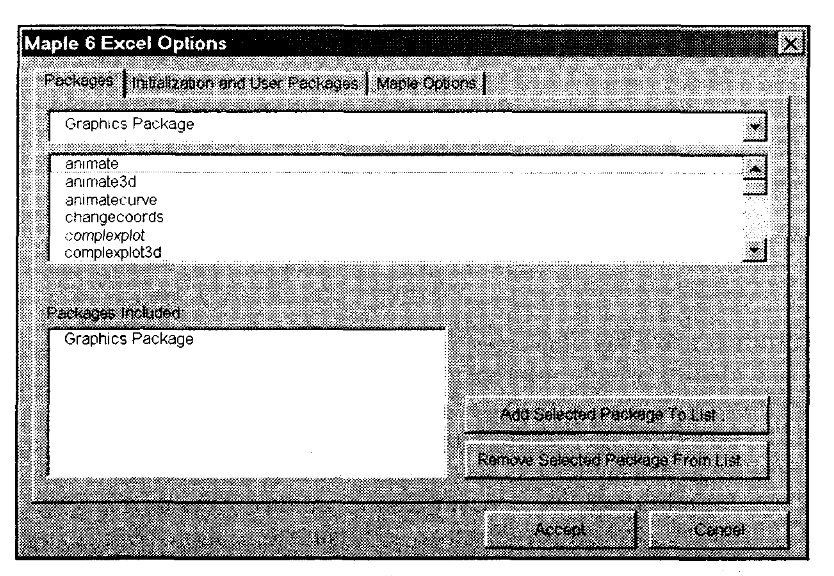


Рис. 6.15. Подключение пакетов Maple в сеансе Excel

В раскрывающемся списке этой вкладки следует выбрать необходимый пакет (при этом в расположенном ниже списке будут отображены все функ-

ции) и нажатием на кнопку **Add Selected Package To List** занести выбранный пакет в список **Packages Included**. Выполнить указанные выше действия для других пакетов, команды которых должны быть доступны в текущем сеансе Excel. После того, как все необходимые пакеты будут занесены в список **Packages Included**, нажатием кнопки **Accept** следует выполнить подключение выбранных пакетов.

Если в дальнейшем для выполнения вычислений команды какого-либо пакета будут не нужны, следует снова отобразить диалоговое окно **Maple 6 Excel Options**, выделить в списке подключенных пакетов необходимый пакет и нажатием кнопки **Remove Selected Package From List** удалить его из списка. Кнопка **Accept**, как всегда, служит для подтверждения и выполнения предпринятых изменений.

Внимание!

Процедуру подключения/отключения пакетов можно выполнять сколько угодно раз в течение сеанса Excel. По его завершении будет сохранена информация о последнем списке подключенных пакетов и они будут загружены в начале следующего сеанса Excel.

Как и в случае с подключением пакетов в Maple, если происходит переопределение каких-либо команд и функций, то эта информация отображается в диалоговом окне сообщений как при загрузке Excel, так и всякий раз, когда пользователь подключает подобный пакет во время сеанса Excel.

Замечание

Названия пакетов в списке не соответствуют названиям аналогичных пакетов в самом Maple, например, пакет Basic Graphical Objects соответствует пакету `plottools` приложения Maple. Однако их наименования в Excel подобраны так, чтобы пользователь мог быстро сообразить, для каких целей предназначены его команды, тем более, что всегда можно просмотреть список содержащихся в пакете программ. Получить соответствие названий пакетов в Excel и "родных" Maple-наименований можно получить только программным способом через VBA *(см. ниже раздел 6.4)*.

На вкладке **Initialization and User Packages** (рис. 6.16) задается пользовательский файл инициализации Maple, который выполняется дополнительно к системному файлу инициализации maple.ini и в котором обычно содержатся команды установки значений системных переменных, например, `Digits` и `Order`, необходимых для выполнения вычислений во время всего сеанса Maple. В поле **Maple Initialization File** следует задать полное имя файла инициализации или кнопкой **Browse** отобразить стандартное окно открытия файлов приложений MS Office и в нем выбрать требуемый файл.

Кроме файла инициализации на этой вкладке можно указать папки с пользовательскими библиотеками Maple (файлы с расширением lib), которые следует подключить на время выполнения сеанса Excel. Для этого следует

нажатием кнопки **Add Package To List** отобразить диалоговое окно открытия файлов, в котором выделить файл библиотеки пользователя и нажать кнопку **Открыть**. Полное имя папки, содержащей выбранный библиотечный файл, автоматически отобразится в списке **Packages Included**, а сама библиотека будет подключена для использования. Удалить библиотеку из списка подключенных библиотек можно кнопкой **Remove Selected Package From List**, предварительно выделив в списке каталогов подключенных библиотек требуемый каталог. Для того чтобы внесенные изменения вступили в силу, следует нажать кнопку **Accept**.

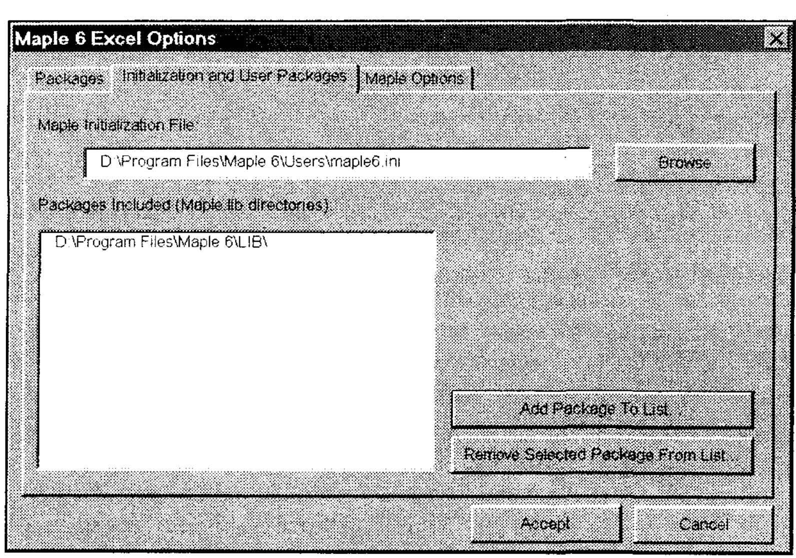


Рис. 6.16. Вкладка задания файла инициализации и библиотек пользователя

Последняя вкладка **Maple Options** позволяет установить значения некоторых параметров Maple для выполнения им определенных действий (рис. 6.17).

В группе **Cancel Calculation Dialog** разрешается или запрещается отображение диалогового окна (флажок **Enabled**), с помощью которого можно прервать выполнение команды Maple, а также задается интервал времени в секундах (поле **Displayed After**), через который после начала выполнения команды это окно отобразится. Появление данного окна, вид которого показан на рис. 6.18, не останавливает вычисление команды Maple. Если нажать кнопку **Yes**, то выполнение команды будет немедленно завершено аварийно, если нажать кнопку **No**, то окно скроется, а команда будет продолжать выполняться. Если пользователь не нажмет ни одной кнопки в этом диалоговом окне, то оно автоматически скроется по завершении выполнения команды Maple. Это окно позволяет пользователю Excel выполнить действие, ассоциированное с кнопкой **Stop** основной панели инструментов приложения Maple.

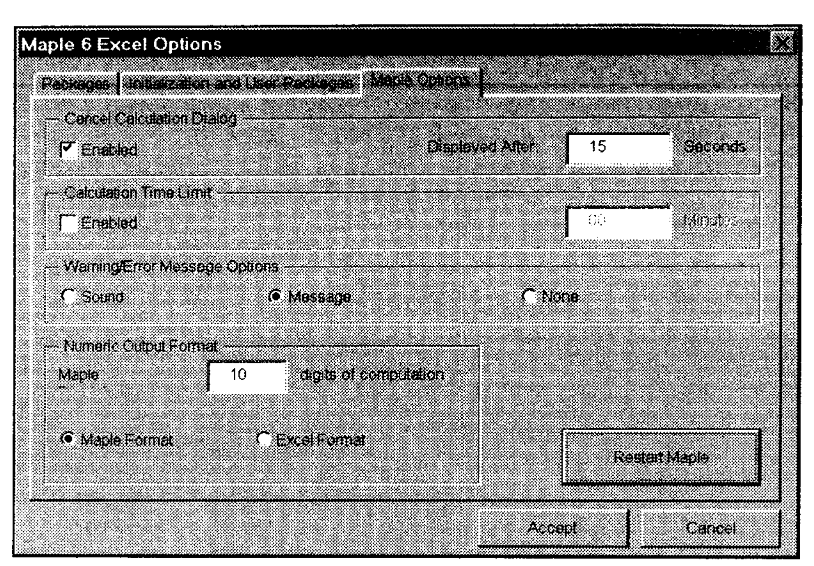


Рис. 6.17. Вкладка задания значений параметров для специальных действий Maple

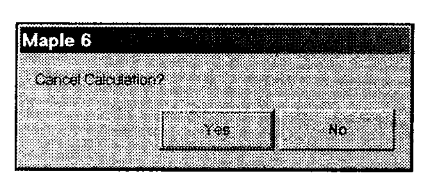


Рис. 6.18. Окно прерывания выполнения команды Maple

В группе **Calculation Time Limit** устанавливается ограничение на время выполнения одной команды. По умолчанию ограничение на время выполнения команды не установлено.

В группе **Warning/Error Message Options** с помощью переключателей выбирается способ информирования пользователя о возникших ошибках при выполнении команды: звуковой сигнал и текст сообщения в ячейке (переключатель **Sound**), текст сообщения в ячейке вместе со звуковым сигналом и отображением диалогового окна с ошибкой (переключатель **Message**) или только текст сообщения в ячейке (переключатель **None**).

Группа **Numeric Output Format** предназначена для задания количества значащих цифр в мантиссе чисел при вычислениях с использованием ядра Maple, а также, в каком формате числовые результаты отображаются в ячейках рабочего листа. Формат Maple отображает числа как текстовые строки с заданным количеством значащих цифр в мантиссе, которые нельзя использовать в вычислениях Excel, тогда как режим, использующий формат Excel, преобразует полученные числа во внутренний формат Excel с возможной потерей точности, но полученные числа можно использовать в вычислениях на рабочем листе.

Действие кнопки **Restart Maple**, последнего элемента управления на этой вкладке, аналогично действию команды `restart` на рабочем листе Maple: все переменные становятся неопределенными, а все подключенные пакеты отключаются.

Внимание!

После нажатия кнопки **Restart Maple** в русской версии MS Excel станет невозможным подключение ни одного пакета Maple. Для того чтобы это действие снова стало доступным, рекомендуется закрыть приложение Excel и снова его загрузить. После выполнения этих действий станет возможным подключение любого другого пакета.

6.4. Программирование функций Maple в VBA

Предыдущие разделы описывали интерактивную работу с командами Maple в Excel. Если же необходимо автоматизировать некоторые действия, следует воспользоваться встроенным во все приложения MS Office, и в том числе в MS Excel, средством программирования VBA (Visual Basic for Applications). Мы предполагаем, что читатель знаком с основами создания приложений на VBA в среде MS Office. Если это не так, то рекомендуем прочитать книгу "Microsoft Office 2000: разработка приложений" издательства BHV, написанную коллективом авторов под редакцией Федора Новикова.

Самый простой способ выполнить команду Maple — задать в ячейке формулу с обращением к функции `Maple()`, что будет в точности соответствовать интерактивной процедуре использования Maple. В примере 6.1 представлен текст процедуры VBA, задающий в ячейке `A1` обращение к построению графика функции, определенной в ячейке `B1`. Имя независимой переменной определено в ячейке `C1`.

Пример 6.1. Занесение в ячейку формулы с вызовом функции Maple

```
Public Sub MapleGraphics()
       Cells(1, 2).Value = "sin(x)*ln(x)
       Cells(1, 3).Value = "x"
       Cells(1, 1).Formula = "=Maple(""plot(&1,&2=0..Pi)",B1,C1)
End Sub
```

Следует сделать два замечания относительно процедуры `MapleGraphics()` примера 6.1. Первое, для задания двойных кавычек в строке VBA их следует повторить два раза. Второе, в качестве разделителя при перечислении ячеек, на которые определены ссылки в команде Maple, использовалась запятая, а

не точка с запятой, как мы делали при непосредственном вводе обращения к функциям Maple в ячейках рабочего листа (это относится к русской версии Excel 2000). Однако если мы после выполнения процедуры `MapleGraphics()` нажатием клавиши <F5> просмотрим на содержимое ячейки `A1`, то обнаружим, что введенные нами разделители-запятые заменены точками с запятой.

Внимание!

При задании параметров функции `Maple()` в коде VBA в качестве разделителя в списке ячеек, на значения которых есть ссылки в команде Maple, определяемой первым параметром, следует использовать символ запятая (`,`).

Такое простое повторение в коде VBA интерактивной процедуры вызова команд Maple в некоторых случаях может оказаться недостаточно эффективным. VBA позволяет непосредственно обращаться к процедурам, реализующим функциональность надстройки Maple 6, и переменным, влияющим на значения устанавливаемых опций ядра Maple. Для этого следует использовать возможности VBA для организации ссылок на открытые процедуры и переменные других проектов.

При подключении надстройки Maple создается проект `Maple6Project`, который можно видеть в окне проектов редактора VBA. Этот проект содержит процедуры, реализующие функциональность надстройки Maple 6, а также определения и значения всех используемых глобальных переменных. Чтобы получить доступ ко всем процедурам и переменным проекта `Maple6Project`, следует задать ссылку на него из текущего проекта в окне **References — VBAProject**, отображаемом командой **Tools ➢ References** редактора VBA. После этого можно просмотреть доступные процедуры и переменные этого проекта в окне просмотра объектов **Object Browser**.

В процедурах VBA вызвать методы проекта `Maple6Project` или получить/установить значения его свойств можно, используя их полные имена:

```
Maple6Project.процедура
Maple6Project.переменная
```

Непосредственное обращение к методам проекта `Maple6Project` позволяет создавать более эффективные алгоритмы с использованием команд Maple и переменных VBA.

Пример 6.2. Обращение к методам проекта Maple6Project

```
Public Sub MapleCalculus()
        Maple6Project.Maple ("Digits:=20")
        f = "sin(x)
        fDer = Maple6Project.Maple("diff(" & f & ",x))
        fInt = Maple6Project.Maple("int(" & CStr(fDer) & ",x=0..Pi/4))
```

```
        fIntNum = Maple6Project.Maple("evalf(%))
        Debug.Print f, fDer, fInt, fIntNum
End Sub
```

Процедура примера 6.2 демонстрирует использование переменных для хранения результатов выполнения команд Maple. Функция `Maple6Project.Maple()` реализует вызов и выполнение команд Maple. Ее первым параметром является строка, содержащая вызываемую команду Maple, а последующие параметры являются объектами `Range`, представляющими ячейки, значения которых подставляются вместо конструкций `&n` в вызове команды. Например, если в ячейке `B1` хранится математическая запись функции в виде алгебраического выражения, а ячейка `C1` содержит имя независимой переменной, то следующий оператор вычислит производную функции по заданной переменной:

```
fDer = Maple6Project.Maple("diff(&1,&2), Cells(1, 2), Cells(1, 3))
```

Замечание

В семействе `Cells` хранятся объекты `Range`, представляющие ячейки рабочего листа Excel. Первый индекс указывает номер ряда, а второй — номер столбца, на пересечении которых расположена ячейка.

Процедура `MapleCalculus()` вычисляет производную функции $\sin(x)$, имя которой хранится в переменной `f`, и сохраняет ее в переменной `fDer`. Кроме этого, она вычисляет точное значение определенного интеграла этой же функции на интервале `[0,π/4]` (переменная `fInt`) и его приближенное значение с двадцатью значащими цифрами в мантиссе вещественного числа (переменная `fIntNum`). Результат отображается в окне отладки редактора VBA.

Кроме использованной в примере 6.2 процедуры `Maple()` в проекте `Maple6Project` содержатся другие доступные из кода процедуры, реализующие основные функции надстройки Maple 6. Перечень всех процедур представлен в табл. 6.1.

Таблица 6.1. *Процедуры проекта* `Maple6Project`

Процедура	Описание
`InitializeGlobals()`	Инициализирует глобальные переменные
`Maple(expr As String, ParamArray ranges() As Variant)`	Выполняет команду Maple, заданную параметром `expr`, с использованием значений в ячейках, определяемых последующими параметрами
`MapleCopyToClipboard()`	Копирует данные из выделенного непрерывного диапазона ячеек рабочего листа в Буфер обмена для последующей вставки в рабочий лист Maple

Таблица 6.1 (окончание)

Процедура	Описание
`MaplePasteToClipboard()`	Вставляет скопированные из электронной таблицы Maple в Буфер обмена данные в рабочий лист Excel
`SetInterruptThreshold(threshold As String)`	Устанавливает интервал времени в секундах, через который отображается окно, позволяющее прервать выполнение команды Maple
`VbRestartMaple()`	Реализует команду `restart()` приложения Maple
`ViewHelp()`	Отображает диалоговое окно **Maple 6 Help** справочной системы Maple
`ViewOptions()`	Отображает диалоговое окно **Maple 6 Excel Options** задания опций и параметров Maple
`ViewWizard()`	Отображает мастера построения команды Maple

Кроме обращения к процедурам проект `Maple6Project` предоставляет возможность устанавливать значения опций ядра Maple, а также получать их текущие значения с помощью специальных переменных. Их имена вместе с кратким описанием назначения представлены в табл. 6.2.

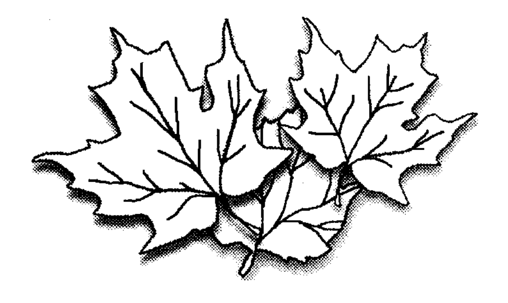
Таблица 6.2. *Глобальные переменные проекта* `Maple6Project`

Переменная	Описание
`ApplicationName`	Хранит имя приложения "Maple 6"
`CurrentWarningType`	Устанавливает тип сообщения об ошибках. Значение `0` — сообщение печатается в ячейке, выполнение команды из которой произошло с ошибкой; `1` — печатает сообщение в ячейке и выдает звуковой сигнал; `2` — печатает сообщение, выдает звуковой сигнал и отображает диалоговое окно с сообщением об ошибке. Для всех трех значений можно использовать встроенные константы `WARNING_NONE`, `WARNING_MSG_SOUND` и `WARNING_SOUND`, соответственно
`PackageArray(0 To 37, 0 To 1)`	Хранит имена подключаемых пакетов, как они отображаются в списке диалогового окна подключаемых пакетов (второй индекс равен `0`), и имя соответствующих им пакетов в приложении Maple (второй индекс равен `1`). Например, элемент с индексом (2,0) имеет значение "Basic Graphical Objects", а элемент с индексом (2,1) — "plottools"

Таблица 6.2 (окончание)

Переменная	Описание
PlotCount	Возвращает количество рисунков Maple на рабочем листе
TimeLimit	Устанавливает/возвращает интервал времени в минутах, по прошествии которого выполнение команды будет прервано ядром Maple
TimeLimitEnabled	Включает (True) или отключает (False) режим прерывания выполнения команды по прошествии интервала времени, определенного в переменной TimeLimit
ValidMaple	Разрешает (True) или запрещает (False) доступ к кнопкам панели инструментов Maple. Если включен запрещающий режим, то нажатие кнопок панели не приводит к выполнению ассоциированных с ними команд

Часть II



Математика

Глава 7. Аналитическая геометрия и линейная алгебра

Глава 8. Дифференцирование функций

Глава 9. Интегрирование функций

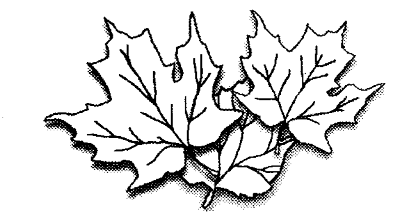
Глава 10. Ряды и дифференциальные уравнения

Глава 11. Численно-аналитические методы

Во второй части книги мы покажем, как можно эффективно использовать Maple для решения задач из общего курса высшей математики, читаемого в большинстве технических университетов. Многие задачи решаются обращением к соответствующей команде Maple, например, вычислить производную функции или интеграл можно, выполнив, соответственно, команды `diff()` или `int()` с подходящими параметрами, но в этой части такой подход нас не будет интересовать. Здесь мы постарались показать, какую помощь может оказать Maple студенту при изучении курса высшей математики, а также как может использовать его преподаватель в процессе обучения. Поэтому вычисление математических объектов осуществляется только на основе их определения или опирается на выведенные в курсе специальные правила. Например, правило интегрирования по частям при вычислении неопределенных интегралов или вычисление определителя матрицы разложением по строке и т. п. Такой подход освобождает студента от рутинных вычислений и в то же время способствует лучшему усвоению и закреплению пройденного материала.

В этой и следующей частях мы в основном будем давать решения типовых задач соответствующих разделов высшей математики с использованием уже известных нам команд Maple, но иногда, как в случае с задачами аналитической геометрии, мы кратко опишем некоторые пакеты Maple, использование команд которых может быть полезным при решении тех или иных задач.

ГЛАВА 7



Аналитическая геометрия и линейная алгебра

Аналитическая геометрия, как на евклидовой плоскости, так и в трехмерном пространстве, является продолжением школьного курса элементарной геометрии, но в отличие от последней изучает геометрические объекты с помощью уравнений, которыми они могут быть описаны, причем порядок этих уравнений не превышает двух. На плоскости это прямая, эллипс, парабола и гипербола, а в пространстве — прямая, плоскость, эллипсоид, гиперболоид, параболоид и др. Изучение высшей математики в технических университетах начинается именно с аналитической геометрии, которую, собственно говоря, можно рассматривать как подготовительную дисциплину для линейной алгебры, изучающей абстрактные линейные пространства и квадратичные формы на них, аналогом которых могут служить кривые и поверхности второго порядка аналитической геометрии.

7.1. Аналитическая геометрия на плоскости

Начнем с задач на построение уравнений прямых линий на плоскости. Одна из типовых задач формулируется так:

Задача 7.1

Уравнение одной из сторон квадрата $x+3y-5=0$. Составить уравнения трех остальных сторон квадрата, если точка с координатами (-1,0) является точкой пересечения его диагоналей.

Решение 1. Будем строить решение задачи с помощью векторов Maple, определяемых командой `vector()`. Этими объектами можно представлять не только векторы на плоскости и в пространстве, но и точки этих пространств. Уравнения линий на плоскости будем хранить и получать в форме уравнений Maple.

Прежде всего определим одну известную сторону квадрата (`line1`) и точку пересечения его диагоналей (`center_point`), являющуюся центром квадрата:

```
> with(linalg):
  line1:=x+3*y-5=0;
  center_point:=vector(2,[-1,0]);
```

$$line1 := x + 3\,y - 5 = 0$$

$$center_point := [-1, 0]$$

Расстояние от точки пересечения диагоналей квадрата до любой его стороны равно половине длины стороны квадрата. Один из способов его определения заключается в определении точки пересечения стороны квадрата с перпендикуляром, опущенном из точки пересечения диагоналей. Для задания прямой линии на плоскости необходимо знать направление перпендикулярного к ней вектора (**a**) и точку ($\mathbf{p}_0$), через которую она проходит. Тогда уравнение прямой может быть записано в виде:

$$(\mathbf{p}-\mathbf{p}_0,\ \mathbf{a})=0,$$

где **p** радиус-вектор текущей точки прямой, а через (·,·) обозначено скалярное произведение двух векторов.

Вектор, перпендикулярный прямой, проходящей через центр квадрата и перпендикулярный заданной его стороне, определяется через уравнение этой стороны. Вектор `[1,3]` перпендикулярен стороне квадрата, а тогда вектор `[-3,1]` перпендикулярен нашей искомой прямой:

```
> perp_vect1:=vector(2,[1,3]);
```

$$perp_vect1 := [1, 3]$$

```
> perp_vect:=vector(2,[3,-1]);
```

$$perp_vect := [3, -1]$$

```
> with(linalg):
  dotprod(perp_vect1,perp_vect);
```

$$0$$

Для проверки перпендикулярности двух векторов мы использовали тот факт, что их скалярное произведение должно быть равно нулю. Команда `dotprod()` пакета `linalg` вычисляет скалярное произведение двух векторов.

Зададим радиус-вектор текущей точки прямой как вектор с координатами `[x,y]`:

```
> point_line:=vector(2,[x,y]);
```

$$point_line := [x, y]$$

Тогда уравнение прямой, перпендикулярной к стороне квадрата, определяется следующим образом:

```
> line:=dotprod((point_line-center_point),perp_vect)=0;
```

$$line := 3\,x + 3 - y = 0$$

Точку пересечения (cross_point) перпендикуляра (line) со стороной квадрата (line1) получим из совместного решения уравнений этих двух прямых:

```
> s:=solve({line,line1},{x,y});
```

$$s := \{x = \frac{-2}{5}, y = \frac{9}{5}\}$$

```
> cross_point:=vector(2,[eval(x,s),eval(y,s)]);
```

$$cross_point := \left[\frac{-2}{5}, \frac{9}{5}\right]$$

Для дальнейших построений нам необходимо вычислить расстояния между двумя точками плоскости. Для выполнения этого действия мы разработали специальную процедуру distance(), двумя параметрами которой являются векторы, представляющие координаты двух точек плоскости:

```
> distance:=proc(v1::vector,v2::vector)
      local i,s;
      s:=0;
      if `linalg/vectdim`(v1) <> `linalg/vectdim`(v2) then
            error("Векторы %1 и %2 должны быть одинаковой \
            размерности", v1,v2);
      end if;
      for i from 1 to `linalg/vectdim`(v1) do
            s:=s+(v1[i]-v2[i])^2;
      end do;
      simplify(sqrt(s));
      end:
```

Теперь можно определить расстояние между точкой центра квадрата и точкой пересечения перпендикуляра из центра на сторону квадрата:

```
> d:=distance(cross_point,center_point);
```

$$d := \frac{3}{5}\sqrt{10}$$

Эта величина равна половине длины стороны квадрата, а так как точка cross_point как раз и делит сторону квадрата пополам. Теперь мы легко можем определить две вершины квадрата v1 и v2 как точки прямой заданной стороны квадрата и расположенные по обе стороны от точки cross_point на расстоянии d:

```
> zero:=vector(2,[0,0]);
```

$$zero := [0, 0]$$

```
> v1:=vector(2,[cross_point[1]+perp_vect[1]/distance(perp_vect,zero)*d,
              cross_point[2]+perp_vect[2]/distance(perp_vect,zero)*d]);
```

$$v1 := \left[\frac{7}{5}, \frac{6}{5}\right]$$

```
> v2:=vector(2,[cross_point[1]-perp_vect[1]/distance(perp_vect,zero)*d,
              cross_point[2]-perp_vect[2]/distance(perp_vect,zero)*d]);
```

$$v2 := \left[\frac{-11}{5}, \frac{12}{5}\right]$$

При вычислении вершин квадрата нам пришлось использовать единичный направляющий вектор прямой line1, построенный из вектора perp_vect (он параллелен стороне квадрата) делением его координат на модуль этого вектора, который вычисляется как расстояние от точки, представленной координатами вектора, до начала координат (zero).

Теперь, зная координаты двух вершин квадрата, можно построить уравнения еще двух его сторон (line2 и line4), а также вычислить координаты остальных двух вершин (v4 и v3), смещаясь вдоль построенных прямых сторон квадрата от соответствующих вершин (v1 и v2) на расстояние 2*d, равное длине стороны квадрата:

```
> line2:=dotprod((point_line-v1),perp_vect)=0;
```

$$line2 := 3x - 3 - y = 0$$

```
> line4:=dotprod((point_line-v2),perp_vect)=0;
```

$$line4 := 3x + 9 - y = 0$$

```
> v4:=vector(2,[v1[1]-perp_vect1[1]/distance(perp_vect1,zero)*2*d,
              v1[2]-perp_vect1[2]/distance(perp_vect1,zero)*2*d]);
```

$$v4 := \left[\frac{1}{5}, \frac{-12}{5}\right]$$

```
> v3:=vector(2,[v2[1]-perp_vect1[1]/distance(perp_vect1,zero)*2*d,
              v2[2]-perp_vect1[2]/distance(perp_vect1,zero)*2*d]);
```

$$v3 := \left[\frac{-17}{5}, \frac{-6}{5}\right]$$

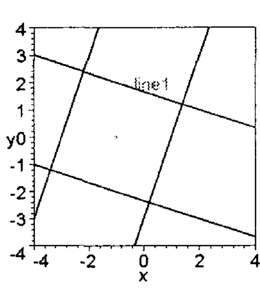
Получить уравнение последней стороны квадрата не представляет труда. Для этого достаточно построить уравнение прямой, проходящей через точку v4 и параллельной заданной стороне квадрата (у них одинаковый перпендикулярный вектор):

```
> line3:=dotprod((point_line-v4),perp_vect1)=0;
```

$$line3 := x + 7 + 3y = 0$$

Для проверки построенных уравнений сторон квадрата можно начертить прямые линии сторон и точку центра квадрата, чтобы убедиться в правильности построенного решения:

```
> with(plots):
> f:=plot({rhs(isolate(line1,y)),
          rhs(isolate(line2,y)),
          rhs(isolate(line3,y)),
          rhs(isolate(line4,y))},x=-4..4,y=-4..4,
          color=black,scaling=CONSTRAINED,thickness=2):
  g:=plot([[center_point[1],center_point[2]]],x=-4..4,
          style=POINT,color=black):
  t:=textplot([1,2,"line1"],color=red,align={LEFT}):
  display({f,g,t},axes=BOXED);
```



Решение 2. В Maple включен специальный пакет `geometry` для выполнения построений на евклидовой плоскости, который можно использовать для решения задач аналитической геометрии.

В этом пакете с помощью специальных команд определяются геометрические объекты: точка (команда `point()`), линия (команда `line()`), отрезок (команда `segment()`), направленный отрезок (команда `dsegment()`), окружность (команда `circle()`), парабола (команда `parabola()`), гипербола (команда `hyperbola()`), эллипс (команда `ellipse()`), квадрат (команда `square()`) и треугольник (команда `triangle()`). Все команды определения геометрических объектов первым параметром задают имя объекта, по которому можно на него в дальнейшем ссылаться, остальные параметры зависят от типа создаваемого объекта и соответствуют способам определения соответствующих объектов. Например, задать окружность можно и с помощью уравнения, и с помощью трех точек, и с помощью точки центра и радиуса:

```
> with(geometry):
> point(p1,[-1,0]); point(p2,[0,1]); point(p3,[1,0]); point(p0,[0,0]);
```

$$p1$$

$$p2$$

$$p3$$

$$p0$$

```
> circle(c1,x^2+(y)^2=1,[x,y]);
```

$$c1$$

```
> circle(c2,[p1,p2,p3],[x,y]);
```

$c2$

```
> circle(c3,[p0,1],[x,y]);
```

$c3$

Здесь все три команды `circle()` определяют одинаковые окружности единичного радиуса с центром в начале координат. Третий параметр во всех командах задания объектов, которые можно описать с помощью уравнений, представляет двухэлементный список наименований независимых переменных. Если его не задавать, то Maple будет отображать подсказки для ввода соответствующих наименований координат, после которых следует ввести идентификатор, завершающийся точкой с запятой:

```
> circle(c1,x^2+y^2=1);
enter name of the horizontal axis > x;
enter name of the vertical axis > y;
```

$c1$

Замечание

Здесь полужирным шрифтом выделена информация, которую пользователь вводит в ответ на сообщения Maple.

Для каждого объекта определен ряд команд, вычисляющих некоторые его характеристики. Так, для объекта окружность можно вычислить координаты центра (`center(имя_окружности)`) и радиус (`radius(имя_окружности)`), для точки ее координаты (`coordinates(имя_точки)`) и т. д. В пакете есть команды, применяемые практически ко всем геометрическим объектам и определяющие общие для всех объектов характеристики:

- `form()` — возвращает тип геометрического объекта: `point2d`, `segment2d`, `dsegment2d`, `line2d`, `triangle2d`, `square2d`, `circle2d`, `ellipse2d`, `parabola2d`, `hyperbola2d` или `FAIL`, если не возможно определить тип геометрического объекта;
- `Equation()` — уравнение геометрического объекта;
- `HorizontalName()` — возвращает имя независимой горизонтальной координаты;
- `VerticallName()` — возвращает имя независимой вертикальной координаты;
- `detail()` — возвращает подробную информацию о геометрическом объекте.

Приведем несколько примеров использования перечисленных команд к определенным выше точкам и окружностям:

```
> detail(c2);
```

name of the object: c2

form of the object: circle2d

name of the center: center_c2

coordinates of the center: [0, 0]

radius of the circle: 1

equation of the circle: x^2+y^2-1 = 0

```
> detail(p0);
```

name of the object: p0

form of the object: point2d

coordinates of the point: [0, 0]

```
> form(c2);
```

circle2d

Для визуализации геометрических объектов пакета `geometry` предназначена команда `draw()`, первым параметром которой задается множество объектов, подлежащих отображению на рисунке Maple, а последующие параметры представляют собой известные опции графических команд в форме уравнений, действие которых распространяется на все отображаемые объекты. Для каждого объекта в множестве после его имени в скобках можно определить опции, применяемые только к этому геометрическому объекту, однако список допустимых опций ограничивается следующими опциями: `color`, `linestyle`, `numpoints`, `style`, `symbol`, `thickness`, `printtext` и `filled`. Установка значения опции `printtext` равным `true` приводит к отображению имен точек, а опция `filled=true`, примененная к замкнутой кривой (окружность, эллипс, квадрат), закрашивает ее внутренность заданным в опции `color` цветом. Пример 7.1 демонстрирует использование команды `draw()`.

Пример 7.1. Отображение геометрических объектов пакета `geometry`

```
> circle(c1,x^2+(y-2)^2=4,[x,y],'centername'=o);
```

c1

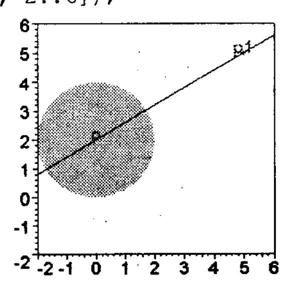
```
> point(p1,[5,5]);
```

p1

```
> line(l1,[o,p1],[x,y]);
```

l1

```
> draw({c1(filled=true,color=green),p1,l1(color=black)},printtext=true,
        view=[-2..6,-2..6]);
```



В этом же пакете определены некоторые команды построения геометрических объектов относительно других, например, перпендикулярной прямой к существующей, а также разнообразные преобразования объектов плоскости — перенос, поворот, подобие и т. д. Со всеми этими и другими командами пакета geometry можно подробнее ознакомиться на странице справки, отображаемой командой ?geometry.

После краткого знакомства с пакетом geometry можно перейти к решению нашей задачи. Прежде всего создадим точку пересечения диагоналей квадрата (p) и прямую линию, на которой расположена заданная сторона квадрата (l1):

```
> point(p,[-1,0]);
```

$$p$$

```
> line(l1,x+3*y-5=0,[x,y]);
```

$$l1$$

Спроецируем точку пересечения диагоналей p на прямую l1 и найдем расстояние между полученной точкой и точкой пересечения диагоналей, которое равняется половине длины стороны квадрата:

```
> projection(p1,p,l1);
```

$$p1$$

```
> coordinates(p1);
```

$$\left[\frac{-2}{5}, \frac{9}{5}\right]$$

```
> d:=distance(p,p1);
```

$$d := \frac{1}{5}\sqrt{18}\,\sqrt{5}$$

Теперь, как и в первом решении, построим две вершины квадрата, расположенные на прямой l1 по разные стороны от точки проекции центра квадрата на расстоянии d от нее:

```
> point(v1,[HorizontalCoord(p1)+d*3/sqrt(10),
          VerticalCoord(p1)-d*1/sqrt(10)]);
```

$$v1$$

```
> point(v2,[HorizontalCoord(p1)-d*3/sqrt(10),
          VerticalCoord(p1)+d*1/sqrt(10)]);
```

$$v2$$

Теперь можно восстановить перпендикуляры из полученных точек вершин квадрата к прямой линии, на которой расположена его заданная в условиях задачи сторона:

```
> PerpendicularLine(l2, v1, l1);
```

$$l2$$

```
> PerpendicularLine(l3, v2, l1);
```

$$l3$$

Проведем диагонали, проходящие через точки вершин v1 и v2, и отобразим построенные линии и точки:

```
> line(d2,[v2,p]);
```

d2

```
> line(d1,[v1,p]);
```

d1

```
> draw({l1,l2,l3,d1,d2,p,p1,v1,v2},printtext=true);
```

Из рисунка видно, что мы на правильном пути, и для завершения решения задачи следует найти точки пересечения диагоналей с соответствующими линиями сторон квадрата:

```
> intersection(v3,d1,l3);
```

v3

```
> intersection(v4,d2,l2);
```

v4

Завершим задачу построением линии четвертой стороны и отображением полученного квадрата с промежуточными построениями:

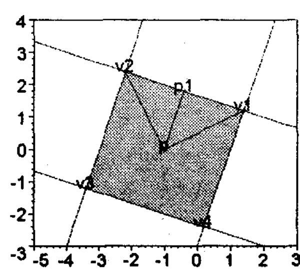
```
> line(l4,[v3,v4]);
```

l4

```
> s:={p,v1,v2,v3,v4,l1,l2,l3,l4};
```

$$s := \{p, l1, l2, l3, v1, v2, v3, v4, l4\}$$

```
> draw(s union {square(sq1,[v1,v2,v3,v4])(filled=true,color=green),
      p1,segment(s1,[p,v2]),segment(s2,[p,v1]),segment(s3,[p,p1])},
      printtext=true,color=blue,view=[-5..3,-3..4]);
```



Анализ рисунка показывает, что построенный нами квадрат является именно тем, который требуется построить по условиям нашей задачи. Остается только распечатать уравнения всех его сторон:

```
> Equation(l1);
```

$$-5 + x + 3\,y = 0$$

```
> simplify(Equation(l2));
```

$$-3 + 3\,x - y = 0$$

```
> simplify(Equation(l3));
```

$$9 + 3\,x - y = 0$$

```
> simplify(Equation(l4));
```

$$\frac{42}{5} + \frac{6}{5}x + \frac{18}{5}y = 0$$

```
> simplify(5*Equation(l4)/6);
```

$$7 + x + 3\,y = 0$$

Для получения приемлемого вида уравнения линии l4 нам пришлось умножить его на 5 и разделить на 6.

Все полученные уравнения совпадают с соответствующими уравнениями сторон квадрата, полученными в первом варианте решения задачи.

Задача 7.2

Составить уравнение линии, каждая точка которой равноотстоит от оси ординат и от окружности $x^2+y^2=4x$.

Решение. Сначала разработаем алгоритм решения. Чтобы построить уравнение искомой линии, необходимо вычислить расстояние от произвольной точки плоскости до произвольной окружности и приравнять его значению горизонтальной координаты точки (это как раз и будет расстояние от точки до оси ординат). Расстояние от точки до окружности равняется абсолютной величине разности расстояния между точкой и центром окружности и радиусом окружности. (Абсолютная величина берется потому, что точка может лежать как вне, так и внутри окружности.) Чтобы не связываться с абсолютной величиной для получения уравнения кривой, каждая точка которой равно удалена как от окружности, так и от оси ординат, будем приравнивать квадраты соответствующих расстояний.

Построим решение с использованием векторов и команд пакета geometry. Прежде всего создадим окружность и определим ее радиус:

```
> restart:with(geometry):
> circle(c1,x^2+y^2=4*x,[x,y],'centername'=o1);
```

$$c1$$

```
> r:=radius(c1);
```

$$r := \sqrt{4}$$

Теперь зададим в форме вектора текущую точку кривой с координатами (x,y), точку центра окружности, вычислим расстояние между ними и определим квадрат расстояния от произвольной точки плоскости до заданной окружности:

```
> p:=vector(2,[x,y]);
```

$$p := [x, y]$$

```
> o:=vector(2,[coordinates(o1)[1],coordinates(o1)[2]]);
```

$$o := [2, 0]$$

```
> d:=(sqrt((p[1]-o[1])^2+(p[2]-o[2])^2)-r)^2;
```

$$d := (\sqrt{x^2 - 4\,x + 4 + y^2} - \sqrt{4}\,)^2$$

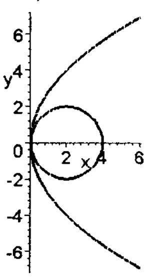
Уравнение искомой линии получим, приравняв величину d квадрату абсциссы точки p:

```
> line:=simplify(isolate(d=p[1]^2,x));
```

$$line := x = \frac{1}{8}y^2$$

При задании линии мы одновременно выразили переменную x в виде функции от переменной y и упростили полученную зависимость, что привело к решению в виде параболы, проходящей через начало координат, ветви которой расположены в первой и четвертой четвертях:

```
> with(plots):
  implicitplot({line,Equation(c1)},x=0..6,y=-10..10,
               scaling=CONSTRAINED,thickness=3,color=black);
```



7.2. Аналитическая геометрия в пространстве

Инструменты Maple для решения задач аналитической геометрии в пространстве мало чем отличаются от использованных нами для решения плоских задач — вместо двумерных векторов следует задавать трехмерные, а вместо команд пакета geometry использовать команды пакета geom3d.

Работа с пакетом geom3d аналогична работе с пакетом geometry — пространственные геометрические объекты создаются с помощью набора специальных команд, первым параметром которых является имя объекта, а остальные должны однозначно определять соответствующий объект. Например, для плоскости можно задать три пространственные точки, точку и ортогональный направленный отрезок, прямую и точку и т. д. Пространственный пакет геометрии поддерживает следующий набор объектов: точка (команда point()), отрезок (команда segment()), направленный отрезок (команда dsegment()), пространственная прямая (команда line()), плоскость (команда plane()), пространственный треугольник (команда triangle()), сфера (команда sphere()) и полиэдр (несколько специальных команд).

Кроме команд создания геометрических объектов пакет поддерживает ряд команд для вычисления характеристик соответствующих объектов: радиуса и центра сферы, координат точки и т. п, а также позволяет выполнять разнообразные преобразования объектов и построения новых объектов, связанных некоторым взаимоотношением с существующими, например, построить прямую линию, перпендикулярную заданной плоскости. Полное описание пакета geom3d можно найти на странице Справки Maple, отображаемой командой ?geom3d.

Задача 7.3

Написать уравнение плоскости, параллельной плоскости $5x-14y+2z+36=0$ и отстоящей от нее на заданном расстоянии d.

Решение 1. Зададим исходную плоскость в виде уравнения:

```
> plain:=5*x-14*y+2*z+36=0;
```

$$plain := 5\,x - 14\,y + 2\,z + 36 = 0$$

Искомую плоскость зададим в виде общего уравнения плоскости с неизвестными коэффициентами:

```
> s:=A*x+B*y+C*z+D=0;
```

$$s := A\,x + B\,y + C\,z + \mathrm{D} = 0$$

Так как искомая плоскость должна быть параллельна заданной, то на основании условия параллельности двух плоскостей можем записать уравнения связи их коэффициентов через произвольный параметр λ:

```
> A:=5*lambda; B:=-14*lambda; C:=2*lambda;
```

$$A := 5\,\lambda$$

$$B := -14\,\lambda$$

$$C := 2\,\lambda$$

В силу алгоритма полной вычислимости выражений в Maple уравнение искомой плоскости после присваивания коэффициентам значений через параметр λ будет иметь вид:

```
> s;
```

$$5\,\lambda\,x - 14\,\lambda\,y + 2\,\lambda\,z + \mathrm{D} = 0$$

Поделим уравнение искомой плоскости на параметр λ:

```
> s:=expand(s/lambda);
```

$$s := 5\,x - 14\,y + 2\,z + \frac{\mathrm{D}}{\lambda} = 0$$

Теперь выберем произвольную точку на искомой плоскости. Если задаться ее двумя координатами x и y, равными нулю, то третья будет равна свободному члену уравнения, деленному на -2:

```
> p:=vector(3,[0,0,-op(4,lhs(s))/2]);
```

$$p := \left[\,0, 0, -\frac{1}{2}\frac{\mathrm{D}}{\lambda}\,\right]$$

Вычислим расстояние от этой точки до заданной в задаче плоскости в соответствии с известной формулой:

```
> m:=sqrt(op(1,op(1,lhs(plain)))^2+
         op(1,op(2,lhs(plain)))^2+
         op(1,op(3,lhs(plain)))^2);
```

$$m := 15$$

```
> dist:=abs(eval(lhs(plain),[x=p[1],y=p[2],z=p[3]]))/m;
```

$$dist := \frac{1}{15}\left|-36 + \frac{\mathrm{D}}{\lambda}\right|$$

Теперь остается приравнять выражение dist величине d и решить полученное уравнение относительно коэффициента D:

```
> res:=solve(dist=d,D);
```

$$res := 36\,\lambda + 15\,d\,\lambda,\ 36\,\lambda - 15\,d\,\lambda$$

Это уравнение имеет два решения. Следовательно, можно построить две плоскости, параллельные заданной и расположенные на заданном расстоянии. Для получения их уравнений необходимо в уравнение искомой плоскости s последовательно подставить эти два решения:

```
> simplify(eval(s,D=res[1]));
```

$$5\,x - 14\,y + 2\,z + 36 + 15\,d = 0$$

```
> simplify(eval(s,D=res[2]));
```

$$5\,x - 14\,y + 2\,z + 36 - 15\,d = 0$$

Данные два уравнения и есть уравнения искомых плоскостей, параллельных заданной в задаче плоскости и расположенных на расстоянии d относительно нее.

Решение 2. Как и в случае с задачей 7.1 из раздела аналитической геометрии на плоскости, найдем решение задачи 7.3 с помощью пакета geom3d геометрических построений в пространстве.

Построим заданную в задаче плоскость, а параллельную ей плоскость зададим посредством уравнения с одним неизвестным коэффициентом — свободным членом, тогда как коэффициенты при независимых переменных будут равны соответствующим коэффициентам заданной в задаче плоскости:

```
> with(geom3d):
> plane(pl1,5*x-14*y+2*z+36=0,[x,y,z]);
```

$$pl1$$

```
> plane(pl2,5*x-14*y+2*z+D=0,[x,y,z]);
```

$$pl2$$

Наша цель найти неизвестный коэффициент D из условия, что расстояние между этими двумя плоскостями равно *d*. Для этого вычислим расстояние между плоскостями pl1 и pl2:

```
> dist:=distance(pl2,pl1);
```

$$dist := \frac{1}{15}|\mathrm{D} - 36|$$

Теперь остается приравнять полученное выражение dist значению d и решить полученное уравнение относительно неизвестной D:

```
> sol:=solve(dist=d,D);
```

$$sol := 36 + 15\ d, 36 - 15\ d$$

Для получения уравнения двух плоскостей, отстоящих от заданной на величину d, следует в уравнение плоскости pl2 вместо коэффициента D подставить найденные значения из последовательности sol. Однако подставить вместо коэффициента D найденные значения, не представляется возможным, так как этот коэффициент защищен в связи с его использованием при задании плоскости pl2. Поэтому мы выберем точку на искомой плоскости pl2, ее координата z будет зависеть от коэффициента D. Вычислив ее координаты командой coordinates(), которая возвращает их в форме списка, подставляя вместо D его вычисленные значения sol, построим плоскости, проходящие через соответствующие точки и параллельные заданной плоскости:

```
> point(p2,[0,0,-D/2]);
```

$$p2$$

```
> p3:=coordinates(p2);
```

$$p3 := \left[0, 0, -\frac{1}{2}\mathrm{D}\right]$$

```
> point(p4,eval(p3,D=sol[1]));
```

$$p4$$

```
> plane(pl3,[p4,NormalVector(pl2)],[x,y,z]);
```

$$pl3$$

```
> Equation(pl3);
```

$$5\,x-14\,y+2\,z+36+15\,d=0$$

```
> point(p4,eval(p3,D=sol[2]));
```

$$p4$$

```
> plane(pl4,[p4,NormalVector(pl2)],[x,y,z]);
```

$$pl4$$

```
> Equation(pl4);
```

$$5\,x-14\,y+2\,z+36-15\,d=0$$

Как видим, получен результат, полностью совпадающий с результатом нашего первого решения.

Задача 7.4

Написать параметрическое уравнение прямой, проходящей через точку $M(1,7,-4)$, перпендикулярно прямым $x=-2+3t$, $y=1+t$, $z=3t$ и $x=1+2t$, $y=8+2t$, $z=-4+5t$.

Решение. Параметрическое уравнение прямой, проходящей через точку M, запишем в виде списка, в котором первый элемент соответствует координате x, второй y, а третий z:

```
> l:=[1+m*t,7+n*t,-4+p*t];
```

$$l:=[1+m\,t,7+n\,t,-4+p\,t]$$

Направляющие векторы искомой прямой и двух заданных определим через векторы Maple:

```
> v:=vector(3,[m,n,p]);
```

$$v:=[m,n,p]$$

```
> v1:=vector(3,[3,1,3]);
```

$$v1:=[3,1,3]$$

```
> v2:=vector(3,[2,2,5]);
```

$$v2:=[2,2,5]$$

Для перпендикулярности искомой кривой необходимо и достаточно, чтобы ее направляющий вектор был перпендикулярен направляющим векторам заданных прямых, а это приводит нас к системе двух уравнений, которые получаются из условия равенства нулю скалярных произведений направляющего вектора искомой прямой и направляющих векторов заданных прямых:

```
> eq1:=`linalg/dotprod`(v,v1)=0;
```

$$eq1:=3\,m+n+3\,p=0$$

```
> eq2:=`linalg/dotprod`(v,v2)=0;
```

$$eq2 := 2\,m + 2\,n + 5\,p = 0$$

```
> s:=solve({eq1,eq2},{m,n});
```

$$s := \{n = -\frac{9}{4}p, m = -\frac{1}{4}p\}$$

```
> assign(s);
> l;
```

$$\left[1 - \frac{1}{4}p\,t, 7 - \frac{9}{4}p\,t, -4 + p\,t\right]$$

Решение системы двух линейных уравнений ищется относительно двух неизвестных переменных m и n, а результат представляется через третью неизвестную p, которая может принимать произвольные значения. Для конкретизации направляющего вектора искомой прямой положим p=4 и построим графики трех прямых:

```
> p:=4;
```

$$p := 4$$

```
> l;
```

$$[1 - t, 7 - 9\,t, -4 + 4\,t]$$

```
> l1:=[-2+3*t,1+t,3*t];
```

$$l1 := [-2 + 3\,t, 1 + t, 3\,t]$$

```
> l2:=[1+3*t,8+2*t,-4+5*t];
```

$$l2 := [1 + 3\,t, 8 + 2\,t, -4 + 5\,t]$$

```
> g:=plot3d({l,l1,l2},t=-10..10,y=-10..10,axes=BOXED):
  f:=`plots/textplot3d`([-10,-60,20,"Искомая прямая"]):
  `plots/display3d`({g,f});
```



Замечание

В этом решении мы не подключали никаких пакетов, а вызывали команды, задавая их полные имена с указанием пакета, причем, так как в этом случае приходится использовать символ "/", то полное имя команды заключается в обратные кавычки.

7.3. Линейная алгебра

Одной из первых задач, решаемых в курсе линейной алгебры, является задача решения системы линейных алгебраических уравнений. Существует много способов ее решения, но классическим считаются два — метод Крамера и метод исключения Гаусса. Первый в настоящее время практически не используется из-за накопления ошибок округления при реальных вычислениях на подмножестве вещественных чисел с ограниченной мантиссой, хотя, справедливости ради, следует заметить, что при вычислениях в Maple подобной проблемы не существует, так как пользователь всегда может увеличить количество цифр в мантиссе используемых чисел, присвоив системной переменной `Digits` требуемое значение. Метод Гаусса широко используется не только в учебных, но и в практических вычислениях, поэтому с него мы и начнем демонстрацию решения задач линейной алгебры.

Задача 7.5

Дана система линейных уравнений

$$\begin{cases} 3x_1 + 2x_2 + x_3 = 5 \\ 2x_1 + 3x_2 + x_3 = 1 \\ 2x_1 + x_2 + 3x_3 = 11 \end{cases}$$

Доказать ее совместность и решить методом Гаусса.

Решение. Неоднородная квадратная система линейных уравнений совместна при любом векторе правой части, если определитель, составленный из ее коэффициентов (он называется определителем системы), не равен нулю. Проверим это условие для нашей системы. Прежде всего определим на рабочем листе матрицу системы `A` и вектор правой части `B`:

```
> A:=matrix(3,3,[[2,1,0],[-3,4,0],[-2,1,2]]);
```

$$A := \begin{bmatrix} 2 & 1 & 0 \\ -3 & 4 & 0 \\ -2 & 1 & 2 \end{bmatrix}$$

```
> B:=vector(3,[5,1,11]);
```

$$B := [5, 1, 11]$$

Вычислить определитель в Maple можно, обратившись к команде `det()` пакета `linalg` или написать рекурсивную процедуру вычисления определителя разложением, например, по первому столбцу. Пример 7.2 содержит текст процедуры вычисления определителя квадратной матрицы.

Пример 7.2. Процедура вычисления определителя квадратной матрицы

```
> det_matrix:=proc(a::matrix)
     local i,m,n,det,c;
```

```
m:=`linalg/rowdim`(a);
n:=`linalg/coldim`(a);
if m <> n then
        error("Матрица %1 должна быть квадратной", a);
end if;
if m = 1 then
        return(a[1,1]);
end if;
det:=0;
for i from 1 to m do
        c:=`linalg/delcols`(`linalg/delrows`(a,i..i),1..1);
        det:=det+(-1)^(1+i)*a[i,1]*det_matrix(c)
end do;
end proc:
```

В первом операторе if процедуры det_matrix осуществляется проверка, является ли переданная через формальный параметр a матрица квадратной. Далее проверяется размерность матрицы — если она равна 1, то процедура завершает свое вычисление, возвращая в качестве определителя значение его единственного элемента. Если размерность больше единицы, то в цикле for вычисляется определитель матрицы разложением по первому столбцу. На каждом шаге цикла строится матрица c вычеркиванием *i*-й строки и первого столбца исходной матрицы a, определитель которой называется минором элемента $a_{i,1}$ и обозначается $M_{i,1}$, и по правилу разложения определителя по столбцу вычисляется сумма

$$\sum_{i=1}^{m} (-1)^{(i+1)} a_{i,1} M_{i,1},$$

которая и является определителем матрицы a. Вычисление минора осуществляется рекурсивным обращением к разработанной нами процедуре det_matrix().

Теперь можно вычислить определитель матрицы A системы уравнений задачи:

```
> det_matrix(A);
```

22

Замечание

Наша процедура вычисления определителя матрицы det_matrix() работает без подключения пакета linalg, так как в ее теле все команды этого пакета вызываются с использованием их полных имен.

Можно (даже необходимо) проверить полученный результат вычислением определителя матрицы с помощью команды det() пакета linalg:

```
> with(linalg):det(A);
```

22

Как и следовало ожидать, получен такой же результат, что и с помощью процедуры det_matrix(). Определитель квадратной системы линейных уравнений не равен нулю, и следовательно, система имеет единственное решение.

По условию задачи получить ее решение следует с помощью метода исключения Гаусса. В Maple нет специальной команды решения системы линейных уравнений этим методом, поэтому нам предстоит разработать собственную процедуру. В примере 7.3 приведен текст процедуры gauss_solve() решения системы линейных уравнений методом исключения Гаусса.

Пример 7.3. Решение системы линейных уравнений методом исключения Гаусса

```
> gauss_solve:=proc(a::matrix,b::vector)
      local i,j,k,c,r,x,m;
      if `linalg/rowdim`(a) <> `linalg/vectdim`(b) then
            error("Количество столбцов матрицы %1 должно равняться \
                  размерности вектора %2",a,b);
      elif `linalg/rowdim`(a) <> `linalg/coldim`(a) then
            error("Матрица %1 должна быть квадратной", a);
      end if;
      m:=`linalg/rowdim`(a);
      c:=a;
      c:=`linalg/concat`(c,b);
      x:=vector(m);
# Прямой ход
      for i from 1 to m do
            r:=1/c[i,i];
            c:=`linalg/mulrow`(c,i,r);
            if i <> m then
                  for j from i+1 to m do
                        r:=-c[j,i];
                        c:=`linalg/addrow`(c,i,j,r);
                  end do;
            end if;
      end do;
# Обратный ход
      x[m]:=c[m,m+1];
      for i from m-1 to 1 by -1 do
            x[i]:= c[i,m+1]-sum(x[k]*c[i,k],k=(i+1)..m);
      end do;
      eval(x);
      end proc:
```

В прямом ходе метода расширенная матрица системы c (матрица системы с присоединенным вектором правой части) приводится к верхнему тре-

угольному виду с единицами на главной диагонали. В обратном ходе в локальном векторе x процедуры вычисляется вектор решения системы, который и является ее возвращаемым значением. При приведении матрицы к верхнетреугольному виду в двойном цикле используется команда addrow() пакета linalg, а при последовательном вычислении элементов вектора решения — команда sum(). Эти команды реализуют циклы, которые при разработке в другом языке программирования, например, Fortran или С, пришлось бы программировать явным образом.

Итак, у нас все готово для решения системы уравнений — ее матрица и вектор правой части заданы, процедура метода исключения Гаусса разработана, и остается только получить решение:

```
> sol:=gauss_solve(A,B);
```

$$sol := \left[\frac{19}{11}, \frac{17}{11}, \frac{71}{11}\right]$$

Полученное решение следует обязательно проверить:

```
> evalm(A&*sol);evalm(B);
```

$$[5, 1, 11]$$
$$[5, 1, 11]$$

Вычислив произведение матрицы системы A на полученный вектор решения sol, видим, что результирующий вектор полностью совпадает с вектором правой части, а значит найденное решение верное.

Для проверки можно воспользоваться и командой linsolve() пакета linalg, которая решает не только квадратную систему линейных уравнений, но и общую систему линейных уравнений с числом переменных, не обязательно равным числу уравнений. Первые два параметра этой команды представляют матрицу системы и вектор правой части, третий параметр является именем переменной, в которой будет содержаться по завершении выполнения команды ранг матрицы системы:

```
> linsolve(A,B,'res');
```

$$\left[\frac{19}{11}, \frac{17}{11}, \frac{71}{11}\right]$$

```
> res;
```

$$3$$

Иногда возникает необходимость отображения преобразованной матрицы системы на каждом шаге метода Гаусса. Это можно реализовать, добавив в нашу процедуру третий параметр, управляющий печатью промежуточных значений — если он равен true, то промежуточные результаты печатаются, если равен false, то нет. Текст подобной процедуры приведен в примере 7.4.

Пример 7.4. Решение системы линейных уравнений с промежуточной печатью

```
> gauss_solve_print:=proc(a::matrix,b::vector,step::boolean)
       local i,j,k,c,r,x,m;
       if `linalg/rowdim`(a) <> `linalg/vectdim`(b) then
              error("Количество столбцов матрицы %1 должно равняться \
              размерности вектора %2",a,b);
       elif `linalg/rowdim`(a) <> `linalg/coldim`(a) then
              error("Матрица %1 должна быть квадратной", a);
       end if;
       m:=`linalg/rowdim`(a);
       c:=a;
       c:=`linalg/concat`(c,b);
       x:=vector(m);
# Прямой ход
       for i from 1 to m do
              r:=1/c[i,i];
              c:=`linalg/mulrow`(c,i,r);
              if i <> m then
                     for j from i+1 to m do
                            r:=-c[j,i];
                            c:=`linalg/addrow`(c,i,j,r);
                     end do;
              end if;
              if step then print(eval(c)) end if;  # Добавлен
       end do;
# Обратный ход
       x[m]:=c[m,m+1];
       for i from m-1 to 1 by -1 do
              x[i]:= c[i,m+1]-sum(x[k]*c[i,k],k=(i+1)..m);
       end do;
       eval(x);
       end proc:
```

Результаты построения решения нашей задачи с печатью промежуточных результатов представлены ниже:

```
> sol:=gauss_solve_print(A,B,true);
```

$$\begin{bmatrix} 1 & \frac{1}{2} & 0 & \frac{5}{2} \\ 0 & \frac{11}{2} & 0 & \frac{17}{2} \\ 0 & 2 & 2 & 16 \end{bmatrix}$$

$$\begin{bmatrix} 1 & \frac{1}{2} & 0 & \frac{5}{2} \\ 0 & 1 & 0 & \frac{17}{11} \\ 0 & 0 & 2 & \frac{142}{11} \end{bmatrix}$$

$$\begin{bmatrix} 1 & \frac{1}{2} & 0 & \frac{5}{2} \\ 0 & 1 & 0 & \frac{17}{11} \\ 0 & 0 & 1 & \frac{71}{11} \end{bmatrix}$$

$$sol := \left[\frac{19}{11}, \frac{17}{11}, \frac{71}{11}\right]$$

При вызове модифицированной процедуры `gauss_solve_print()` с третьим параметром `true` мы можем видеть матрицы с последовательно исключенными переменными. Если третий параметр будет равен `false`, то модифицированная процедура будет работать аналогично процедуре `gauss_solve()` примера 7.3.

Таким образом, в процессе решения исходной задачи мы показали все возможные подходы получения решения систем линейных алгебраических уравнений в Maple — и с помощью пользовательской процедуры, и с помощью команд встроенных пакетов.

Следующая задача будет связана с линейными преобразованиями n-мерных пространств. Известно, что любое линейное преобразование однозначно определяется своей матрицей в некотором базисе

$$y = Ax$$

Здесь y и x представляют n-мерные векторы двух пространств, а матрица A является матрицей линейного преобразования одного пространства в другое.

Задача 7.6

Даны два линейных преобразования $x' = A\,x$ и $x'' = B\,x'$ трехмерных пространств с матрицами

$$A = \begin{bmatrix} 4 & 3 & 5 \\ 6 & 7 & 1 \\ 9 & 1 & 8 \end{bmatrix}$$

$$B = \begin{bmatrix} -1 & 3 & -2 \\ -4 & 1 & 2 \\ 3 & -4 & 5 \end{bmatrix}$$

Найти преобразование, выражающее вектор x'' через вектор x.

Решение. Матрица результирующего преобразования является произведением матрицы *B* на матрицу *A*. Здесь мы также можем либо воспользоваться возможностями Maple для выполнения произведения двух матриц, либо разработать собственную процедуру, что мы и сделали. Текст процедуры перемножения двух прямоугольных матриц представлен в примере 7.5.

Пример 7.5. Умножение двух прямоугольных матриц

```
> mat_mul:=proc(a::matrix,b::matrix)
      local i,j,k,c,m,n,l;
      if `linalg/coldim`(a) <> `linalg/rowdim`(b) then
            error("Количество столбцов матрицы %1 должно равняться \
            количеству строк матрицы %2",a,b)
      end if;
      m:= `linalg/rowdim`(a);
      l:= `linalg/coldim`(a);
      n:= `linalg/coldim`(b);
      c:=matrix(m,n);
      for i from 1 to m do
            for j from 1 to n do
                  c[i,j]:=sum(a[i,k]*b[k,j],k=1..l)
            end do;
      end do;
      eval(c);
      end proc:
```

В этой процедуре `mat_mul()` прежде всего проверяется, имеет ли входная матрица `a` такое же количество столбцов, что и матрица `b`, так как только при этом условии определена операция умножения двух прямоугольных матриц. Если это условие выполняется, то далее в двойном цикле по количеству строк матрицы `a` и количеству столбцов матрицы `b` с помощью команды `sum()` вычисляется произведение строки `i` матрицы `a` на столбец `j` матрицы `b` и полученный результат присваивается элементу $c_{i,j}$ матрицы произведения.

Теперь с помощью разработанной нами процедуры `mat_mul()` решение исходной задачи не представляет труда. Определяем на рабочем листе матрицы двух заданных преобразований и перемножаем их:

```
> A:=matrix(3,3,[[4,3,5],[6,7,1],[9,1,8]]);
```

$$A := \begin{bmatrix} 4 & 3 & 5 \\ 6 & 7 & 1 \\ 9 & 1 & 8 \end{bmatrix}$$

```
> B:=matrix(3,3,[[-1,3,-2],[-4,1,2],[3,-4,5]]);
```

$$B := \begin{bmatrix} -1 & 3 & -2 \\ -4 & 1 & 2 \\ 3 & -4 & 5 \end{bmatrix}$$

```
> mat_mul(A,B);
```

$$\begin{bmatrix} -1 & -5 & 23 \\ -31 & 21 & 7 \\ 11 & -4 & 24 \end{bmatrix}$$

Для проверки полученного результата можно перемножить матрицы с помощью операции `&*`, не забыв использовать команду `evalm()`:

```
> evalm(A&*B);
```

$$\begin{bmatrix} -1 & -5 & 23 \\ -31 & 21 & 7 \\ 11 & -4 & 24 \end{bmatrix}$$

или воспользоваться командой `multiply()` пакета `linalg`, в которой можно задать произвольное количество матриц, и она вычислит их последовательное произведение в заданном порядке:

```
> multiply(A,B);
```

$$\begin{bmatrix} -1 & -5 & 23 \\ -31 & 21 & 7 \\ 11 & -4 & 24 \end{bmatrix}$$

Последняя задача линейной алгебры, которую мы решим, будет задача о вычислении собственных значений и соответствующих им собственных векторов линейного преобразования, заданного своей матрицей в некотором базисе.

Задача 7.7

Найти собственные значения и соответствующие им собственные векторы линейного преобразования, заданного в некотором базисе матрицей

$$A = \begin{bmatrix} 0 & 1 & 0 \\ -3 & 4 & 0 \\ -2 & 1 & 2 \end{bmatrix}$$

Решение. Прежде всего следует найти собственные числа матрицы линейного преобразования из решения его характеристического уравнения

$$|A - \lambda E| = 0,$$

в котором E является единичной матрицей, а $|\ |$ означает определитель.

Составим характеристическое уравнение, предварительно задав саму матрицу A задачи и единичную матрицу E:

```
> with(linalg):
> A:=matrix(3,3,[[0,1,0],[-3,4,0],[-2,1,2]]);
```

$$A := \begin{bmatrix} 0 & 1 & 0 \\ -3 & 4 & 0 \\ -2 & 1 & 2 \end{bmatrix}$$

```
> E:=array(1..3,1..3,identity);
```

$$E := \text{array}(\,identity, 1 \,..\, 3, 1 \,..\, 3, [\ ]\,)$$

```
> DD:=det(A-l*E);
```

$$DD := -11\, l + 6\, l^2 - l^3 + 6$$

Переменная `DD` содержит характеристический полином матрицы `A`. Отметим, что его можно получить командой `charpoly()` пакета `linalg`:

```
> DD:=charpoly(A,'l');
```

$$DD := l^3 - 6\, l^2 + 11\, l - 6$$

Замечание

Команда `charpoly()` строит упорядоченный по степеням переменной, определяемой ее вторым параметром, характеристический полином матрицы, задаваемой в качестве первого параметра этой команды.

После того как характеристический полином построен, следует найти его нули, которые и будут являться собственными числами матрицы (линейного оператора):

```
> lambda:=solve(DD,l);
```

$$\lambda := 1, 2, 3$$

В нашей задаче матрица линейного оператора имеет три не равных между собой собственных числа, а это говорит о том, что три соответствующих им собственных вектора, являясь линейно независимыми, образуют базис этого оператора, и в данном базисе матрица оператора будет диагональной с собственными числами, расположенными на главной диагонали.

Для вычисления собственного вектора, соответствующего собственному числу λ_i, следует решить систему линейных уравнений

$$(A - \lambda_i E)\, x = 0,$$

в которой 0 означает нулевой вектор пространства. Мы решим в цикле три подобные системы:

```
> B:=vector(3,[0,0,0]);
```

$$B := [0, 0, 0]$$

```
> for i to 3 do
      b[i]:=linsolve(A-lambda[i]*E,B,'r','v');
  end do;
```

$$b_1 := [v_1, v_1, v_1]$$
$$b_2 := [0, 0, v_1]$$
$$b_3 := [v_1, 3\,v_1, v_1]$$

Проверим, что найденные нами решения, правильные. Для этого вычислим последовательно произведения Ab_i и посмотрим, равняются ли они, соответственно, векторам $\lambda_i b_i$:

```
> for i to 3 do
      evalm(A&*b[i])=evalm(lambda[i]*b[i]);
  end do;
```

$$[v_1, v_1, v_1] = [v_1, v_1, v_1]$$
$$[0, 0, 2\,v_1] = [0, 0, 2\,v_1]$$
$$[3\,v_1, 9\,v_1, 3\,v_1] = [3\,v_1, 9\,v_1, 3\,v_1]$$

Поверка показала, что найденные решения верны.

Присвоив произвольной переменной v_1 значение 1, можно получить конкретный набор нормированных собственных векторов, являющихся базисом заданного в задаче линейного оператора:

```
> v[1]:=1:
  for i to 3 do
      expand([b[i][1],b[i][2],b[i][3]]/sqrt(sum(b[i][k]^2,k=1..3)));
  end do;
```

$$\left[\frac{1}{3}\sqrt{3}, \frac{1}{3}\sqrt{3}, \frac{1}{3}\sqrt{3}\right]$$
$$[0, 0, 1]$$
$$\left[\frac{1}{11}\sqrt{11}, \frac{3}{11}\sqrt{11}, \frac{1}{11}\sqrt{11}\right]$$

Мы решили задачу определения собственных значений и собственных векторов линейного оператора, заданного своей матрицей в некотором базисе, определив корни характеристического уравнения с помощью команды `solve()`, однако заметим, что система Maple содержит ряд команд, находящихся в пакете `linalg`, предназначенных для определения спектра матриц. К ним относятся команды `eigenvalues()` для вычисления собственных значений матрицы и `eigenvectors()`, рассчитывающая не только собственные числа матрицы, но одновременно и соответствующие им собственные векторы:

```
> eigenvalues(A);
```

$$1, 2, 3$$

```
> eigenvectors(A);
```

$$[1, 1, \{[1, 1, 1]\}], [3, 1, \{[1, 3, 1]\}], [2, 1, \{[0, 0, 1]\}]$$

Команда `eigenvalues()` вычисляет собственные значения матрицы и возвращает их в виде последовательности. Команда `eigenvectors()` возвращает последовательность списков, каждый из которых содержит три элемента: собственное число, его кратность и множество собственных векторов, являющихся базисом подпространства, соответствующего собственному числу, размерности не более кратности собственного числа:

```
> A := matrix([[1, 2, 0], [-3, 8, 0], [-2, 1, 2]]);
```

$$A := \begin{bmatrix} 1 & 2 & 0 \\ -3 & 8 & 0 \\ -2 & 1 & 2 \end{bmatrix}$$

```
> eigenvalues(A);
```

$$7, 2, 2$$

```
> eigenvectors(A);
```

$$[2, 2, \{[0, 0, 1]\}], [7, 1, \{[5, 15, 1]\}]$$

В данном примере размерность подпространства, соответствующего собственному числу 2 матрицы A и имеющего кратность 2, равняется 1, а поэтому и множество собственных векторов этого собственного числа состоит из единственного вектора.

Представленный нами способ вычисления собственных векторов и собственных чисел линейного оператора (матрицы), а также упомянутые команды позволяют производить вычисления и с матрицами, элементами которых представлены алгебраическими выражениями:

```
> A := matrix(2,2, [a,-3,-6,0]);
```

$$A := \begin{bmatrix} a & -3 \\ -6 & 0 \end{bmatrix}$$

```
> e := eigenvalues(A);
```

$$e := \frac{1}{2}a + \frac{1}{2}\sqrt{a^2+72}, \frac{1}{2}a - \frac{1}{2}\sqrt{a^2+72}$$

```
> v := [eigenvectors(A)];
```

$$v := \left[\left[\frac{1}{2}a + \frac{1}{2}\sqrt{a^2+72}, 1, \left\{\left[-\frac{1}{12}a - \frac{1}{12}\sqrt{a^2+72}, 1\right]\right\}\right],\right.$$
$$\left.\left[\frac{1}{2}a - \frac{1}{2}\sqrt{a^2+72}, 1, \left\{\left[-\frac{1}{12}a + \frac{1}{12}\sqrt{a^2+72}, 1\right]\right\}\right]\right]$$

Для вычисления собственных чисел и векторов числовых матриц можно воспользоваться отложенной командой `Eigenvals()`, расположенной в ядре Maple. С ее помощью можно найти приближенные значения с заданной точностью указанных величин, причем собственные векторы представляют-

ся в виде матрицы, хранящейся в переменной, задаваемой вторым параметром этой команды:

```
> A := matrix([[1, 2, 0], [-3, 8, 0], [-2, 1, 2]]);
```

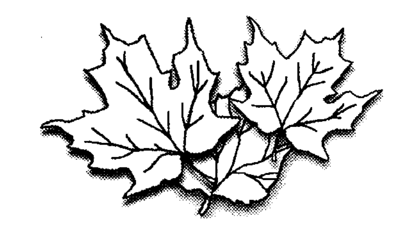
$$A := \begin{bmatrix} 1 & 2 & 0 \\ -3 & 8 & 0 \\ -2 & 1 & 2 \end{bmatrix}$$

```
> evalf(Eigenvals(A,'vecs')); eval(vecs);
```

$$[-2.666042404 + 3.692758675\ I, -2.666042402 - 3.692758674\ I, 1.558582114\ ]$$

$$\begin{bmatrix} 0 & 0. & 1. \\ 0 & -.8660254037 & 1.500000000 \\ 1 & -.8660254037 & .4999999999 \end{bmatrix}$$

ГЛАВА 8



Дифференцирование функций

Производная непрерывной функции является одной из важнейших ее характеристик. Она используется в задаче исследования поведения функции, построения ее графика, в задачах условной и безусловной оптимизации и многих других. Производная непрерывной функции одной переменной в заданной точке *x* определяется как предел отношения ее приращения к приращению аргумента при стремлении последнего к нулю:

$$\lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Для вычисления производных в Maple существует специальная команда `diff()`, которая позволяет вычислять как производные любых порядков функций одной переменной, так и частные производные функций многих переменных. Однако для первоначального знакомства с производной ее лучше определять с помощью предела на основании приведенного определения. Для вычисления пределов в Maple предназначена команда `limit()`, с которой мы и познакомимся в первом разделе этой главы.

8.1. Пределы

Команда вычисления предела имеет две формы, одна из которых `limit()` вычисляет предел, а вторая — отложенная — служит для отображения на рабочем листе и в тексте комментариев математической записи предела и отличается от вычисляемой команды только именем, начинающимся с прописной буквы, — `Limit()`. Общий синтаксис этих команд следующий:

```
limit(f, x=a [,dir]);
Limit(f, x=a [,dir]);
```

Параметр f определяет выражение, зависящее от неизвестной величины x, предел которого необходимо вычислить при стремлении неизвестной к величине a. По умолчанию Maple считает независимую переменную вещественной, а любые параметры в выражении ненулевыми и также вещественными, если не задано иное с помощью команды assume(). Величина a может быть любым выражением с неизвестными величинами, числом (вещественным или комплексным), а также — infinity или infinity. В случае отсутствия параметра dir команда вычисляет обычный предел функции за исключением случаев, когда задано стремление независимой переменной к бесконечности (infinity) или минус бесконечности (-infinity). В этих случаях пределы вычисляются при стремлении переменной x к минус бесконечности справа, а к плюс бесконечности слева. Использование команд вычисления пределов показано в примере 8.1.

Пример 8.1. Вычисление обычных пределов

```
> Limit(5*x/arctan(x),x=0)= limit(5*x/arctan(x),x=0);
```

$$\lim_{x \to 0} 5 \frac{x}{\arctan(x)} = 5$$

```
> limit((a*x^2+1)/(b*x+c),x=infinity);
```

$$\frac{\text{signum}(a)\,\infty}{\text{signum}(b)}$$

```
> limit((a*x^2+1)/(b*x+c),x=d*c);
```

$$\frac{1 + a\,d^2\,c^2}{c\,(b\,d + 1)}$$

Значение параметра dir задает вычисление предела при стремлении переменной x справа к величине a (right) или слева (left). Если его значение равно real, то вычисляется обычный предел функции вещественной переменной, причем в этом случае значение infinity трактуется не как плюс бесконечность, к которой переменная x стремится слева, а как одновременно плюс/минус бесконечность и предел рассматривается в обычном смысле, т. е. если предел при стремлении к плюс бесконечности слева равен пределу при стремлении к минус бесконечности справа, то предел существует и равен полученному значению, в противном случае предела выражения не существует. Значение complex параметра dir определяет, что предел ищется в комплексной области, и его значение не зависит от способа стремления независимой переменной к предельной точке, при этом значение infinity рассматривается как бесконечно удаленная точка расширенной комплексной области. Пример 8.2 демонстрирует использование этого параметра для вычисления односторонних пределов как в вещественной, так и в комплексной областях.

Пример 8.2. Вычисление односторонних пределов

```
> limit(exp(x),x=infinity);
```

$$\infty$$

```
> limit(exp(x),x=-infinity);
```

$$0$$

```
> limit(exp(x),x=infinity, real);
```

$$\mathit{undefined}$$

```
> Limit(1/x, x=0, right)=limit(1/x, x=0, right);
```

$$\lim_{x \to 0+} \frac{1}{x} = \infty$$

```
> Limit(1/x, x=0, left)=limit(1/x, x=0, left);
```

$$\lim_{x \to 0-} \frac{1}{x} = -\infty$$

```
> Limit(1/x, x=0)=limit(1/x,x=0);
```

$$\lim_{x \to 0} \frac{1}{x} = \mathit{undefined}$$

```
> Limit(1/x, x=0, complex)=limit(1/x,x=0, complex);
```

$$\lim_{x \to 0,\mathrm{complex}} \frac{1}{x} = \infty - \infty\, I$$

Обратите внимание на последний предел примера 8.2 — в возвращаемом значении команды `limit()` может использоваться и мнимая единица I, и символ бесконечности ∞.

Завершим этот раздел решением типовых задач на вычисление пределов из курса высшей математики.

Задача 8.1

Найти предел, не пользуясь правилом Лопиталя:

$$\lim_{x \to 0} \frac{1}{3} \frac{\sqrt{1+x} - \sqrt{1-x}}{x}$$

Решение. Определим выражение, предел которого необходимо вычислить:

```
> v:=(sqrt(1+x)-sqrt(1-x))/3/x;
```

$$v := \frac{1}{3} \frac{\sqrt{1+x} - \sqrt{1-x}}{x}$$

В точке $x = 0$ оно имеет неопределенность *0/0*, избавиться от которой можно, умножив и числитель, и знаменатель на одну и ту же величину ($\sqrt{1+x} + \sqrt{1-x}$). В Maple нам придется это действие выполнить за два шага: сначала умножить числитель и упростить его командой `expand()`, а затем уже умножить знаменатель на эту же величину:

```
> v1:=expand(v*(sqrt(1+x)+sqrt(1-x)));
```

$$v1 := \frac{2}{3}$$

```
> v2:=v1/(sqrt(1+x)+sqrt(1-x));
```

$$v2 := \frac{2}{3}\frac{1}{\sqrt{1+x}+\sqrt{1-x}}$$

Полученное, эквивалентное исходному выражение v2 не имеет в точке $x=0$ неопределенности, а поэтому его предел при $x \to 0$ равен просто значению этого выражения в предельной точке:

```
> eval(v2, x=0);
```

$$\frac{1}{3}$$

Проверить полученный результат можно непосредственным выполнением команды `limit()` для исходного выражения v:

```
> limit(v,x=0);
```

$$\frac{1}{3}$$

Как видим, наш результат совпадает с результатом, вычисленным Maple.

Задача 8.2

Для функции f(x) определить, является ли она непрерывной в точке $x=2$:

$$f(x) = 9^{\left(\frac{1}{2-x}\right)}$$

Решение. Вычислим пределы заданной функции справа и слева в точке $x=2$:

```
> y:=x->9^(1/(2-x));
```

$$y := x \to 9^{\left(\frac{1}{2-x}\right)}$$

```
> limit(y(x),x=2,right);
```

$$0$$

```
> limit(y(x),x=2,left);
```

$$\infty$$

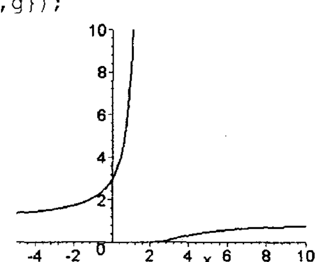
Предел справа не равен пределу слева, а поэтому заданная функция имеет в точке $x=2$ разрыв. Более того, эта функция не определена в данной точке, так как при $x=2$ знаменатель показателя степени обращается в нуль.

Отобразить график этой функции простым обращением к команде `plot()` с ограничением диапазона значений не удается из-за слишком больших значений функции слева от точки разрыва:

```
> plot(y(x),x=1..5, 0..10);
Plotting error, exponent too large
```

Поэтому для построения графика пришлось прибегнуть к искусственному приему, представив график как объединение двух графиков — слева от точки разрыва, причем диапазон изменения независимой переменной не включает саму точку разрыва, и справа от точки разрыва:

```
> f:=plot(y(x),x=-5..1.99,0..10, color=black, thickness=2):
> g:=plot(y(x),x=2.0001..10,0..10, color=black, thickness=2):
> with(plots):display({f,g});
```



Задача 8.3

Задана функция y=f(x). Найти точки разрыва функции, если они существуют. Построить ее график.

$$f(x) = \begin{cases} x+4 & x < -1 \\ x^2+2 & -1-x \le 0 \text{ and } x < 1 \\ 2x & 1 \le x \end{cases}$$

Решение. Зададим кусочно-непрерывную функцию командой `piecewise()`:

```
> y:=piecewise(x<-1,x+4,x>=-1 and x<1,x^2+2,x>=1, 2*x);
```

$$y := \begin{cases} x+4 & x < -1 \\ x^2+2 & -1-x \le 0 \text{ and } x < 1 \\ 2x & 1 \le x \end{cases}$$

Единственные точки, где функция может иметь разрывы, — это границы диапазонов задания кусочно-непрерывной функции, так как внутри диапазонов она задается непрерывными функциями. Для проверки непрерывности следует вычислить пределы справа и слева в точках x=−1 и x=1 (команда `limit()` может работать и с кусочно-непрерывными функциями):

```
> limit(y,x=-1,right);limit(y,x=-1,left);
```

$$3$$

$$3$$

```
> limit(y,x=1,right);limit(y,x=1,left);
```

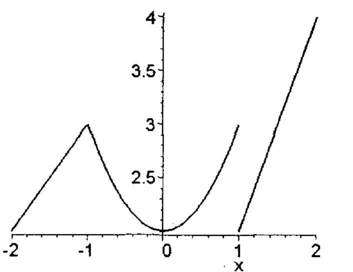
$$2$$

$$3$$

Видим, что в точке $x=-1$ предел справа равен пределу слева, а следовательно, функция в ней непрерывна, чего нельзя сказать относительно точки $x=1$.

График заданной функции построим, как обычно, командой `plot()`:

```
> plot(y,x=-2..2,discont=true,color=black,thickness=2);
```



Задача 8.4

При каких значениях a и b функция y=f(x) будет непрерывна. Построить ее график.

$$f(x)=\begin{cases}(x-1)^2 & x\le 0\\ a\,x+b & -x<0 \text{ and } x<1\\ \sqrt{x} & 1\le x\end{cases}$$

Решение. На заданных интервалах функция непрерывна. Единственными точками, в которых она может иметь разрывы, являются граничные точки интервалов $x=0$ и $x=1$. Для обеспечения непрерывности функции в этих точках необходимо и достаточно, чтобы в них существовали равные между собой пределы функции при стремлении независимой переменной справа и слева к возможной точке разрыва.

Запишем уравнение равенства пределов справа и слева для точки $x=0$:

```
> f:=x->piecewise(x<=0,(x-1)^2,x>0 and x<1,a*x+b,x>=1,sqrt(x));
```

$$f := x \to \text{piecewise}\,(x \le 0, (x-1)^2, 0 < x \text{ and } x < 1, a\,x+b, 1 \le x, \sqrt{x}\,)$$

```
> s1:=limit(f(x),x=0, left)=limit(f(x),x=0, right);
```

$$s1 := 1 = b$$

Получили одно уравнение для нахождения двух неизвестных параметров a и b. Второе уравнение получаем, приравнивая пределы справа и слева в точке $x=1$:

```
> s2:=limit(f(x),x=1, left)=limit(f(x),x=1, right);
```

$$s2 := a + b = 1$$

Решая полученную систему двух уравнений находим значения неизвестных параметров a и b, при которых кусочно-непрерывная функция будет непре-

рывна в граничных точках интервалов ее задания, а тем самым и на всей числовой оси:

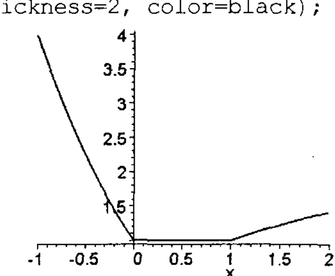
```
> solve({s1,s2}, {a,b});
```

$$\{b=1, a=0\}$$

```
> assign(%);
```

При найденных значениях параметров a=0 и b=1 на интервале (0, 1) функция принимает постоянное значение y=1. Для проверки полученного решения построим график функции с найденными значениями параметров:

```
> plot(f(x),x=-1..2, thickness=2, color=black);
```



8.2. Производная и ее использование для исследования функций

Начнем раздел с демонстрации геометрического смысла производной как тангенса угла наклона касательной к графику функции в заданной точке. Касательная определяется как предельное положение секущей, проходящей через две точки графика при стремлении одной из них к другой. Все построения осуществим на примере функции:

$$y = (e^{\cos(x)} + 3)^2$$

Будем вычислять производную в точке x_0=1. Для этого зададим саму функцию и две точки ее графика с абсциссами, соответственно, x_0=1 и x_1= x_0+h:

```
> y:=x->(exp(cos(x))+3)^2;
```

$$y := x \to (e^{\cos(x)} + 3)^2$$

```
> x0:=1;
```

$$x0 := 1$$

```
> p0:=[x0, y(x0)];
```

$$p0 := [1, (e^{\cos(1)} + 3)^2]$$

```
> p1:=[x0+h, y(x0+h)];
```

$$p1 := [1 + h, (e^{\cos(1+h)} + 3)^2]$$

Команда `slope()` из пакета `student` вычисляет тангенс угла наклона прямой, проходящей через две заданные точки:

```
> with(student): t:=slope(p0,p1);
```

$$t := -\frac{(e^{\cos(1)}+3)^2 - (e^{\cos(1+h)}+3)^2}{h}$$

Теперь, если устремить h к нулю, то выражение `t` должно сходиться к числу, равному тангенсу угла наклона секущей в предельном положение, т. е. тангенсу угла наклона касательной к графику функции в точке x=1. Зададим последовательность значений `h_values`, сходящуюся к нулю, и посмотрим, к чему будет сходиться последовательность значений, определяемых выражением `t`:

```
> h_values:=seq(2/i^3,i=1..15);
```

$$h_values := 2, \frac{1}{4}, \frac{2}{27}, \frac{1}{32}, \frac{2}{125}, \frac{1}{108}, \frac{2}{343}, \frac{1}{256}, \frac{2}{729}, \frac{1}{500}, \frac{2}{1331}, \frac{1}{864}, \frac{2}{2197}, \frac{1}{1372}, \frac{2}{3375}$$

```
> seq(evalf(t), h=h_values);
```



```
> seq(evalf(t), h=h_values);
```


-5.439033250, -12.57034624, -13.3498521, -13.5137879, -13.569058,

-13.592932, -13.604946, -13.611648, -13.615686, -13.61826,

-13.61998, -13.62116, -13.62203, -13.62266, -13.62313

Видно, что эта последовательность сходится, но сходится ли она к значению производной функции в точке x=1? Вычислим производную с помощью функции `diff()`:

```
> evalf(eval(diff(y(x),x),x=1));
```

-13.62516143

Замечаем, что построенная нами последовательность сходится к значению производной в точке x=1. Уже ее пятнадцатый член имеет два точных знака после запятой. Точный результат получим, если вычислим предел выражения `t` при $h \to 0$:

```
> limit(t,h=0);
```

$$-2\,e^{(2\cos(1))}\sin(1) - 6\,e^{\cos(1)}\sin(1)$$

```
> evalf(%);
```

-13.62516143

Графические возможности Maple позволяют увидеть, как секущая приближается к касательной. Построим уравнение секущей как прямой, проходящей через две заданные точки с координатами (`x0`, `y(x0)`) и (`x0+h`, `y(x0+h)`) соответственно (здесь `Y` является зависимой, а `X` независимой переменными):

```
> (Y-y(x0))/(y(x0+h)-y(x0))=(X-x0)/((x0+h)-x0);
```

$$\frac{Y-(e^{\cos(1)}+3)^2}{(e^{\cos(1+h)}+3)^2-(e^{\cos(1)}+3)^2}=\frac{X-1}{h}$$

Выразим зависимую переменную Y через независимую X и представим в виде функции:

```
> isolate(%,Y);
```

$$Y=\frac{(X-1)((e^{\cos(1+h)}+3)^2-(e^{\cos(1)}+3)^2)}{h}+(e^{\cos(1)}+3)^2$$

```
> line_sec:=unapply(rhs(%),X);
```

$$line_sec := X \to \frac{(X-1)((e^{\cos(1+h)}+3)^2-(e^{\cos(1)}+3)^2)}{h}+(e^{\cos(1)}+3)^2$$

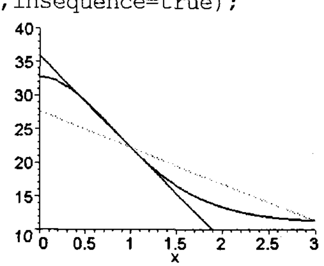
Аналогично построим в виде функции уравнение касательной:

```
> line_tang:=X->eval(diff(y(x),x),x=1)*(X-x0)+y(x0);
```

$$line_tang := X \to \left(\frac{\partial}{\partial x}\,y(x)\right)\Bigg|_{x=1}(X-x0)+y(x0)$$

Теперь можем построить последовательность изображений, содержащих график функции, ее касательной и секущей при изменении параметра h, и отобразить ее в виде анимационной картинки командой display():

```
> S:=seq( plot([y(x), line_tang(x), line_sec(x)], x=0..4,
      view=[0..3,10..40], color=[black,black,green], thickness=2),
      h=h_values):
> with(plots): display(S,insequence=true);
```



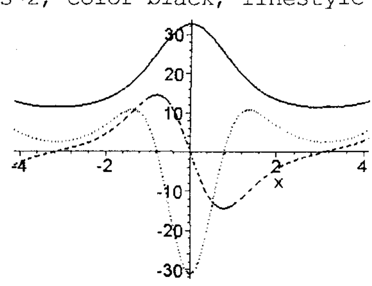
Замечание

Рисунок представляет первый кадр анимационного изображения, на котором график функции и касательной к нему в точке $x=1$ отображаются черными линиями, а секущая серой. При просмотре анимации секущая будет изменять свое положение, приближаясь к касательной и, в конце концов, сливаясь с ней.

С помощью первой и второй производных можно определить характер изменения и поведения исходной функции. В точках равенства нулю первой

производной функция может иметь локальный оптимум, а интервалы ее знакопостоянства определяют диапазоны возрастания и убывания исходной функции, тогда как знак второй производной в подозрительной точке на локальный экстремум определяет, будет ли это максимум (значение отрицательно) и минимум (значение положительно). Точки равенства нулю второй производной функции являются точками перегиба функции, т. е. точки, в которых функция изменяет вогнутость на выпуклость или наоборот. Все это можно наблюдать, если на одном рисунке отобразить графики функции и ее первых двух производных:

```
> plot([y(x),diff(y(x),x),diff(y(x),x$2)],x=-Pi..Pi,
          thickness=2, color=black, linestyle=[1,4,7]);
```



На этом рисунке график функции отображается сплошной линией, первой производной — штриховой, а второй производной — точечной линией. В точке x=0 первая производная равна нулю, и в этой же точке функция имеет локальный максимум, тогда как в точках $x=\pm\pi$, в которых опять-таки первая производная принимает нулевое значение, функция достигает локальных минимумов. В точках $x\approx-0.788$ и $x\approx0.788$ вторая производная обращается в нуль, а функция в этих точках меняет, соответственно, выпуклость на вогнутость и наоборот.

Графическое отображение функции и необходимых производных, конечно, дает представление о характере изменения функции и приблизительных значениях некоторых замечательных точек функции, но математика — наука точная и требует получения точных результатов, тем более что графическое представление ничего нам не скажет о таких знаменательных прямых, как асимптоты. Поэтому в очередной задаче мы покажем, как средствами Maple следует исследовать функцию, а ее график, построенный командой `plot()`, будем использовать только в качестве иллюстрации полученных нами результатов.

Задача 8.5

Исследовать функцию $y=\dfrac{2x-1}{(x-1)^2}$ и построить ее график.

Решение. Будем исследовать функцию в соответствии с общепринятой в математике схемой:

1. Область определения функции и ее характерные точки (граничные, точки разрыва, точки пересечения графика с осями координат).
2. Определение четности, нечетности или периодичности функции.
3. Промежутки знакопостоянства.
4. Промежутки возрастания/убывания и постоянства функции.
5. Локальные экстремумы, наибольшее и наименьшее значения.
6. Промежутки выпуклости/вогнутости функции и точки перегиба.
7. Вертикальные, горизонтальные и наклонные асимптоты.

Если для функции определены все перечисленные величины, то построить график этой функции не представляет никакого труда.

Функция определена на всей числовой оси за исключением точки $x=1$, в которой ее знаменатель обращается в нуль. В этой точке функция имеет разрыв второго рода, так как

```
> Limit(y(x),x=1,left)=limit(y(x),x=1,left);
```

$$\lim_{x \to 1-} \frac{2x-1}{(x-1)^2} = \infty$$

```
> Limit(y(x),x=1,right)=limit(y(x),x=1,right);
```

$$\lim_{x \to 1+} \frac{2x-1}{(x-1)^2} = \infty$$

Найдем корни функции:

```
> y:= x -> (2*x-1)/(x-1)^2;
```

$$y := x \to \frac{2x-1}{(x-1)^2}$$

```
> solve(y(x)=0,x);
```

$$\frac{1}{2}$$

Функция имеет единственный корень в точке `x=1/2`.

Проверим четность/нечетность функции. Для этого необходимо вычислить ее значение при `-x` и сравнить со значением при `x`. Если они равны, то функция четная, если не равны — нечетная:

```
> evalb(y(x)=y(-x));
```

false

```
> evalb(y(x)=-y(-x));
```

false

Функция является функцией общего вида, так как тест на четность/нечетность не прошел, да к тому же функция непериодическая:

```
> solve(y(x)=y(x+T),T);
```

$$0, -2\,\frac{x\,(x-1)}{2\,x-1}$$

Единственное числовое решение уравнения $y(x)=y(x+T)$ равно нулю.

Для определения промежутков знакопостоянства следует решить два неравенства $y\ (x)>0$ и $y\ (x)<0$:

```
> solve(y(x)>0,x);
```

$$\mathrm{RealRange}\left(\mathrm{Open}\left(\frac{1}{2}\right), \mathrm{Open}(1)\right), \mathrm{RealRange}\ (\mathrm{Open}(1), \infty)$$

```
> solve(y(x)<0,x);
```

$$\mathrm{RealRange}\left(-\infty, \mathrm{Open}\left(\frac{1}{2}\right)\right)$$

Записывая результаты решения неравенств в привычной математической нотации, получаем, что функция положительна в области $(1/2,\ 1) \cup (1,\ \infty)$ и отрицательна на интервале $(-\infty,\ 1)$.

Для нахождения участков монотонности функции следует вычислить ее первую производную и решить определить промежутки, где она положительна (на них функция возрастает) и отрицательна (на них функция убывает):

```
> solve(diff(y(x),x)>0,x);
```

$$\mathrm{RealRange}\,(\mathrm{Open}(0), \mathrm{Open}(1))$$

```
> solve(diff(y(x),x)<0,x);
```

$$\mathrm{RealRange}\,(-\infty, \mathrm{Open}(0)), \mathrm{RealRange}\,(\mathrm{Open}(1), \infty)$$

Результаты определения участков монотонности таковы: на промежутке $(0,1)$ функция возрастает, а на интервалах $(-\infty,\ 0)$ и $(1,\ \infty)$ функция убывает.

Чтобы определить подозрительные на экстремум точки, следует приравнять нулю первую производную и найти корни полученного уравнения:

```
> solve(diff(y(x),x)=0,x);
```

$$0$$

Единственная подозрительная на экстремум точка $x=0$. Для определения, является ли эта точка минимумом или максимумом, можно вычислить в ней значение второй производной функции. Если результат будет положительным, то в этой точке минимум, если отрицательным, то максимум, если равен нулю, то следует использовать информацию о промежутках монотонности функции:

```
> eval(diff(y(x),x$2),x=0);
```

$$2$$

Полученное положительное значение говорит о том, что в точке $x=0$ наша функция имеет локальный минимум со значением:

```
> eval(y(x),x=0);
```

$$-1$$

Участки выпуклости и вогнутости определяются по знаку второй производной. Если она положительна, то функция на этом участке вогнута, если отрицательна, то выпукла:

```
> solve(diff(y(x),x$2)>0,x);
```

$$\text{RealRange}\left(\text{Open}\left(\frac{-1}{2}\right), \text{Open}(1)\right), \text{RealRange}(\text{Open}(1), \infty)$$

```
> solve(diff(y(x),x$2)<0,x);
```

$$\text{RealRange}\left(-\infty, \text{Open}\left(\frac{-1}{2}\right)\right)$$

На интервале (-∞, -1/2) функция выпукла, а на интервалах (-1/2,1) и (1, ∞) вогнута.

Чтобы определить точки перегиба, необходимо знать корни уравнения $y''(x)=0$ и проверить, меняет ли вторая производная знак при переходе через них:

```
> solve(diff(y(x),x$2)=0,x);
```

$$\frac{-1}{2}$$

С учетом нахождения участков выпуклости/вогнутости можно утверждать, что точка $x=-1/2$ является точкой перегиба.

Коэффициенты a, b уравнения наклонной асимптоты $y=ax+b$ определяются через вычисление пределов:

$$a = \lim_{x \to \infty} \frac{f(x)}{x} \text{ или } a = \lim_{x \to (-\infty)} \frac{f(x)}{x},$$

$$b = \lim_{x \to \infty} f(x) - ax \text{ или } b = \lim_{x \to (-\infty)} f(x) - ax.$$

Если указанные пределы существуют, то имеются и наклонные асимптоты для соответствующих бесконечных ветвей графика функции (при стремлении x к плюс или минус бесконечности).

Вычислим эти пределы для исследуемой нами функции:

```
> alpha[1]:=limit(y(x)/x, x=+infinity);
```

$$\alpha_1 := 0$$

```
> alpha[2]:=limit(y(x)/x, x=-infinity);
```

$$\alpha_2 := 0$$

```
> a:=alpha[1];
```

$$a := 0$$

```
> beta[1]:=limit(y(x)-a*x, x=+infinity);
```

$$\beta_1 := 0$$

```
> beta[2]:=limit(y(x)-a*x, x=-infinity);
```

$$\beta_2 := 0$$

```
> b:=beta[1];
```

$$b := 0$$

Анализ значений пределов показывает, что у графика исследуемой функции одна горизонтальная асимптота ось абсцисс:

$$y = 0$$

Вертикальными асимптотами для графика функции могут быть только прямые вида $x=x_0$, где x_0 является точкой разрыва второго рода, причем следует проверять пределы при стремлении независимой переменной к точке разрыва второго рода справа и слева. Для нашей функции вертикальной асимптотой будет прямая $x=1$ (см. нахождение области определения функции).

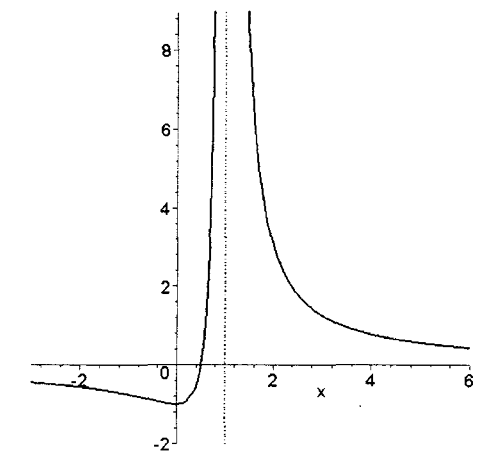
Результаты проведенного исследования сведены в табл. 8.1.

Таблица 8.1. *Результаты исследования функции* $y = \dfrac{2x-1}{(x-1)^2}$

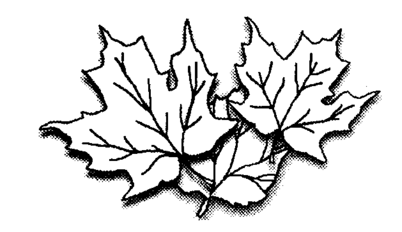
x	**(-∞, -1/2)**	**-1/2**	**(-1/2, 0)**	**0**	**(0, 1)**	**1**	**(1, ∞)**
y'	–	–	–	0	+		–
y''	–	0	+	+	+		+
y	убыв. вып.	-8/9 перегиб	убыв. вогн.	-1 мин.	возр. вогн.	не определена	убыв. вогн.

В табл. 8.1 указаны характерные точки функции, в соответствии с которыми разбита на интервалы область определения функции (вся числовая ось). На каждом интервале указаны знаки первой и второй производных, а также характер изменения (вогнутая/выпуклая, убывает/возрастает) самой функции. В критических точках, входящих в область определения функции, вычислено ее значение и указан тип точки (перегиб или локальный экстремум). Руководствуясь информацией из этой таблицы можно легко построить график исследованной функции. Проверить себя можно построением графика командой `plot()`:

```
> plot([y(x),[1,t,t=-2..9]],x=-3..6,-2..9,
            color=black, thickness=2, linestyle=[1,7]);
```



ГЛАВА 9



Интегрирование функций

В этой главе мы расскажем о том, как вычислять неопределенные интегралы, не прибегая к команде `int()`, а используя только основные правила их вычисления: формулу интегрирования по частям, разложение рациональной дроби на простые, замену переменных. Команды `int()` и `diff()` будет использоваться только для проверки полученного решения.

Также будет продемонстрировано вычисление определенных интегралов через интегральные суммы и их приложение для вычисления числовых характеристик геометрических объектов: длины кривой, площади плоской фигуры, объема тела вращения и т. п.

Мы не приводим никакой теории, ограничиваясь только определениями и формулами, предполагая, что читатель знаком со всеми используемыми нами математическими понятиями и формулами, во всяком случае, он всегда может освежить свои знания, просмотрев любой учебник по математическому анализу.

9.1. Неопределенный интеграл

Неопределенным интегралом функции $\mathrm{f}(x)$ на промежутке X изменения независимой переменной называется множество всех первообразных $\mathrm{F}(x)$ этой функции на заданном промежутке, т. е. функций, удовлетворяющих равенству

$$\mathrm{F}'(x)=\mathrm{f}(x), \quad x \in X.$$

Для обозначения неопределенного интеграла функции используется символ $\int \mathrm{f}(x)\,dx$, при этом

$$\int \mathrm{f}(x)\,dx = \mathrm{F}(x) + C.$$

Эта формула представляет структуру неопределенного интеграла функции: все первообразные отличаются друг от друга на некоторую постоянную величину C.

Одним из наиболее часто применяемых правил при вычислении неопределенных интегралов является правило (или формула) интегрирования по частям:

$$\int u\, dv = u\, v - \int v\, du.$$

В этой формуле u и v являются функциями некоторой независимой переменной.

В пакете student содержится специальная команда intparts(), реализующая формулу интегрирования по частям. Ее использование мы продемонстрируем на примере решения конкретной задачи.

Задача 9.1

Вычислить интеграл $\int e^{(-x)} \cos\left(\frac{1}{2}x\right) dx$.

Решение. Прежде всего подключим пакет student командой

```
>with(student):
```

Для удобства дальнейших ссылок на вычисляемый интеграл присвоим его с помощью отложенной команды Int() переменной Integral:

```
>Integral:=Int(exp(-x)*cos(x/2),x);
```

$$Integral := \int e^{(-x)} \cos\left(\frac{1}{2}x\right) dx$$

Для вычисления интеграла нам придется два раза применить формулу интегрирования по частям. Проинтегрируем по частям исходный интеграл первый раз. Для этого воспользуемся командой intparts(интеграл, u), где первый параметр интеграл является вычисляемым интегралом в виде Int(u*v,x), а второй параметр u представляет функцию u в формуле интегрирования по частям. Чтобы читателю стало яснее, мы записали формулу интегрирования по частям применительно к параметрам команды intparts():

$$\int u\, v\, dx = u \int v\, dx - \int\int v\, dx\, du.$$

Интегрируем исходный интеграл по частям, выбрав $e^{(-x)}$ в качестве функции u:

```
> g:=simplify(intparts(Integral,exp(-x)));
```

$$g := 2\, e^{(-x)} \sin\left(\frac{1}{2}x\right) + 2 \int e^{(-x)} \sin\left(\frac{1}{2}x\right) dx$$

Интегрирование по частям пока не принесло нам желаемого результата. У нас все равно имеется интеграл, который мы не можем тотчас же вычислить, используя таблицу интегралов. Выделим этот интеграл и также применим к нему правило интегрирования по частям:

```
> Integral2:=op(2,g);
```

$$Integral2 := 2 \int e^{(-x)} \sin\left(\frac{1}{2}x\right) dx$$

```
> g1:=simplify(intparts(Integral2,exp(-x)));
```

$$g1 := -4\, e^{(-x)} \cos\left(\frac{1}{2}x\right) - 4 \int e^{(-x)} \cos\left(\frac{1}{2}x\right) dx$$

Видим, что интегрирование по частям интеграла `Integral2` привело к выражению, содержащему исходный неопределенный интеграл. Подставим полученное значение интеграла `Integral2` в переменную `g`, представляющую выражение искомого интеграла `Integral` после первого интегрирования по частям:

```
> g:=subs(Integral2=g1,g);
```

$$g := 2\, e^{(-x)} \sin\left(\frac{1}{2}x\right) - 4\, e^{(-x)} \cos\left(\frac{1}{2}x\right) - 4 \int e^{(-x)} \cos\left(\frac{1}{2}x\right) dx$$

Подведем итог нашим вычислениям. Дважды проинтегрировав по частям, мы пришли к представлению исходного интеграла `Integral` через него же самого. Теперь остается только решить уравнение `Integral=g` относительно неизвестной переменной `Integral` (в правой части уравнения она содержится неявно) для получения значения искомого интеграла:

```
> Integral:=simplify(solve(Integral=g, Integral));
```

$$Integral := -\frac{2}{5}\left(-\sin\left(\frac{1}{2}x\right) + 2\cos\left(\frac{1}{2}x\right)\right) e^{(-x)}$$

Проверить полученный ответ можно дифференцированием `diff(Integral,x)` или вычислением исходного интеграла с помощью команды `int()`:

```
> simplify(diff(Integral,x));
```

$$e^{(-x)} \cos\left(\frac{1}{2}x\right)$$

```
> simplify(int(exp(-x)*cos(x/2),x));
```

$$-\frac{2}{5}\left(-\sin\left(\frac{1}{2}x\right) + 2\cos\left(\frac{1}{2}x\right)\right) e^{(-x)}$$

Обратите внимание на использование команды `simplify()` на протяжении всего примера. Она упрощает получаемые выражения и представляет их в удобном для пользователя виде.

Задача 9.2

Вычислить интеграл $\int \frac{3\,x^2+8}{x^3+4\,x^2+4\,x}\,dx.$

Решение. Подынтегральная функция является правильной рациональной дробью. Вычисление подобных интегралов осуществляется разложением подынтегральной функции на элементарные слагаемые дроби.

Решение начнем с присваивания выражения подынтегральной функции некоторой переменной (dr). Затем выделим из него знаменатель (переменная dn), который разложим на множители и сохраним в переменной dnf:

```
> dr:=(3*x^2+8)/(x^3+4*x^2+4*x);
```

$$dr := \frac{3\,x^2+8}{x^3+4\,x^2+4\,x}$$

```
> dn:=denom(dr);
```

$$dn := x\,(x^2+4\,x+4)$$

```
> dnf:=factor(dn);
```

$$dnf := x\,(x+2)^2$$

Теперь можем "написать" схему разложения подынтегральной дроби на элементарные слагаемые дроби, причем для доступа к структурным элементам выражения dnf будем использовать команду op():

```
> drSimple:=A/op(1,dnf)+B/op(1,op(2,dnf))+C/op(2,dnf);
```

$$drSimple := \frac{A}{x}+\frac{B}{x+2}+\frac{C}{(x+2)^2}$$

Сделаем несколько пояснений относительно использования команды op(). Выражение dnf имеет тип `*` с двумя операторами, первый из которых представляет выражение x, а второй выражение $(x+2)^2$. Поэтому эти два операнда мы используем в первом и третьем слагаемом разложения на элементарные дроби. Для получения первой степени второго сомножителя выражения dnf мы вычисляем первый операнд выражения, представляющего этот второй сомножитель.

В выражении drSimple переменные A, B и C представляют неизвестные величины, которые необходимо определить, приравняв это выражение исходной рациональной дроби dr. Для этого мы приводим к общему знаменателю дробь drSimple:

```
> dns1:=simplify(drSimple);
```

$$dns1 := \frac{A\,x^2+4\,A\,x+4\,A+B\,x^2+2\,B\,x+C\,x}{x\,(x+2)^2}$$

После чего выделяем ее числитель, а также числитель дроби dr:

```
> dns:=numer(dns1);
```

$$dns := A\,x^2 + 4\,A\,x + 4\,A + B\,x^2 + 2\,B\,x + C\,x$$

```
> drs:=numer(dr);
```

$$drs := 3\,x^2 + 8$$

Выделенные числители должны быть равны между собой, так как знаменатели у дробей dns1 и dr одинаковы. Но чтобы они были равными, необходимо, чтобы коэффициенты при одинаковых степенях независимой переменной x были также равными между собой. Это приводит к системе трех уравнений относительно трех неизвестных величин A, B и C в разложении подынтегральной дроби на элементарные дроби. Ее решение завершает процедуру разложения правильной подынтегральной рациональной дроби на элементарные слагаемые дроби:

```
> e1:=coeff(dns,x^2)=coeff(drs,x^2);
```

$$e1 := A + B = 3$$

```
> e2:=coeff(dns,x)=coeff(drs,x);
```

$$e2 := 4\,A + 2\,B + C = 0$$

```
> e3:=coeff(dns,x,0)=coeff(drs,x,0);
```

$$e3 := 4\,A = 8$$

```
> solve({e1,e2,e3},{A,B,C});
```

$$\{A = 2, B = 1, C = -10\}$$

Теперь можно командой assign() присвоить им полученные из решения системы значения и посмотреть на окончательный вид разложения подынтегральной дроби на элементарные слагаемые дроби:

```
> assign(%);
> drSimple;
```

$$2\,\frac{1}{x} + \frac{1}{x+2} - \frac{10}{(x+2)^2}$$

Так как полученное нами выражение является иным представлением исходного подынтегрального выражения, то, естественно, интеграл от него равен интегралу от этого исходного выражения. Теперь остается только вычислить каждый из интегралов суммы drSimple, чтобы получить окончательный ответ.

Но прежде чем переходить к их вычислению, сделаем одно замечание относительно разложения правильной рациональной дроби на элементарные слагаемые дроби. Мы, собственно говоря, повторили процедуру классического алгоритма разложения дроби на сумму элементарных. В Maple можно было бы воспользоваться командой преобразования дроби к типу parfrac, который и представляет собой сумму элементарных дробей для заданной дробно-рациональной функции:

```
> convert(dr,parfrac,x);
```

$$2\frac{1}{x}+\frac{1}{x+2}-\frac{10}{(x+2)^2}$$

Вернемся к вычислению нашего интеграла через полученное разложение. Интеграл от первого слагаемого является табличным, но мы его вычислим с использованием команды `int()`:

```
> In1:=Int(op(1,drSimple),x);
```

$$In1 := \int 2\frac{1}{x}\,dx$$

```
> In1:=value(In1);
```

$$In1 := 2\ln(x)$$

Второй интеграл сводится к табличному простой заменой переменной $u=x+2$. Для автоматизации замены переменных в интеграле можно воспользоваться командой `changevar()` пакета `student`, в которой первым параметром в виде уравнения задается выражение новой переменной интегрирования через старую, а второй параметр представляет сам интеграл:

```
> In2:=Int(op(2,drSimple),x);
```

$$In2 := \int -10\frac{1}{(x+2)^2}\,dx$$

```
> with(student):changevar(x+2=u,In2);
```

$$\int -10\frac{1}{u^2}\,du$$

```
> In2:=value(%);
```

$$In2 := 10\frac{1}{u}$$

Теперь остается только вернуться к старой переменной, используя команду `subs()`:

```
> In2:=subs(u=x+2,In2);
```

$$In2 := 10\frac{1}{x+2}$$

Третий интеграл вычисляется такой же заменой переменной, как и в случае вычисления второго интеграла:

```
> In3:=Int(op(3,drSimple),x);
```

$$In3 := \int \frac{1}{x+2}\,dx$$

```
> In3:=changevar(x+2=u,In3);
```

$$In3 := \int \frac{1}{u}\, du$$

```
> In3:=value(In3);
```

$$In3 := \ln(u)$$

```
> In3:=subs(u=x+2,In3);
```

$$In3 := \ln(x+2)$$

Теперь остается только "записать" исходный интеграл как сумму трех вычисленных интегралов (произвольную постоянную мы опускаем):

```
> Integ:=In1+In2+In3;
```

$$Integ := 2\ln(x) + \frac{10}{x+2} + \ln(x+2)$$

Проверим полученный результат дифференцированием

```
> simplify(diff(Integ,x));
```

$$\frac{3x^2+8}{x(x+2)^2}$$

и интегрирование с помощью команды `int()`

```
> int((3*x^2+8)/(x^3+4*x^2+4*x),x);
```

$$2\ln(x) + \frac{10}{x+2} + \ln(x+2)$$

Проверка показала, что полученный нами результат правильный. Как видно из приведенных решений двух задач, Maple можно использовать не только для быстрого получения результата с помощью подходящих команд, но и для обучения студентов существующим методам интегрирования, причем наличие команд интегрирования по частям и замены переменных всего лишь позволяет ускорить получение результата, ни в коей мере не выступая в роли "умного" устройства, решающего вместо обучающегося задачи. Использование перечисленных команд требует знания основных формул интегрирования, освобождая студента от рутинной работы и позволяя ему мыслить математическими категориями.

В заключение этого раздела мы покажем, как с помощью языка Maple можно написать собственную процедуру интегрирования по частям выражения $f(x)g(x)$, в котором первая функция $f(x)$ представляет собой полином относительно переменной x, а $g(x)$ — функция некоторого специального вида. Например, когда мы вычисляем интеграл от выражения $(x^2 - x + 1)\,e^x$, то результат также представляется в виде произведения некоторого полинома на ту же самую экспоненциальную функцию:

```
> collect(int((x^2-x+1)*exp(x),x),exp(x));
```

$$(x^2 - 3\,x + 4)\,e^x$$

Как мы вычисляем вручную подобные интегралы? Прежде всего разбиваем его на сумму интегралов, общий вид каждого слагаемого в которой можно представить в форме:

$$\int x^n\,e^x\,dx$$

После этого каждый из интегралов интегрируем по частям до тех пор, пока не получим интеграл, вычисляемый непосредственно. Действительно, проинтегрируем представленный выше общий интеграл по частям:

$$\int x^n\,e^x\,dx = x^n\,e^x - n\int x^{(n-1)}\,e^x\,dx$$

Мы видим, что в результате получается новый интеграл, в котором степень переменной x уменьшена на единицу по сравнению с исходным. Повторяя далее эти вычисления по частям для вновь получаемых интегралов, мы дойдем до интеграла от экспоненты, который равен самой экспоненте. Таким образом, вычисление нашего исходного интеграла завершится. При большей степени n нам придется выполнить больше интегрирования по частям, при меньшей меньше, но всегда мы будем завершать вычисления, доходя до интеграла от экспоненты.

Эти размышления можно использовать для создания рекурсивной процедуры вычисления рассматриваемого интеграла, которая при n=0 завершает рекурсию, возвращая экспоненциальную функцию:

```
> IntXnExp:=proc(n::nonnegint, x::name)
              if n=0 then
                    return exp(x)
              else
                    x^n*exp(x)-n*IntXnExp(n-1,x);
              end if;
              end proc:
```

Чтобы вычислить, например, интеграл $\int x^4\,e^x\,dx$, можно вызвать разработанную процедуру со следующими параметрами:

```
> IntXnExp(4,x);
```

$$x^4\,e^x - 4\,x^3\,e^x + 12\,e^x\,x^2 - 24\,e^x\,x + 24\,e^x$$

Теперь на основе разработанной процедуры интегрирования одного члена полинома, умноженного на экспоненциальную функцию, можно написать процедуру интегрирования произведения полинома любой степени на экспоненциальную функцию:

```
> IntPolynomExp:=proc(p::polynom, x::name)
            local i, result;
            result:=add(coeff(p,x,i)*IntXnExp(i,x),
            i=0..degree(p,x));
            collect(result, exp(x));
            end proc:
```

В этой процедуре с помощью команды `add()` вычисляется сумма коэффициентов полинома, умноженных, соответственно, на интеграл $\int x^i e^x dx$, т. е. строится первообразная полинома, умноженного на экспоненту. Интеграл вычисляется с помощью рекурсивной процедуры `IntXnExp()`, разработанной нами ранее. Возвращаемое значение представляет собой полином, умноженный на экспоненту, которое получается в результате выполнения команды `collect()`. Теперь мы можем быстро вычислять интегралы для функций рассмотренного класса с помощью разработанной нами процедуры:

```
> IntPolynomExp((x^3-2)*(x+1),x);
```

$$(18 - 20x - 3x^3 + 9x^2 + x^4)e^x$$

9.2. Приложения определенного интеграла

Напомним, что определенным интегралом функции y=f(x) по промежутку [a,b] называется конечный предел интегральных сумм $\sum_{i=1}^{n} f(\xi_i)\Delta x_i$ при $n \to \infty$, где предел берется по всем разбиениям промежутка [a,b] на n отрезков длиной Δx_i, причем максимальный отрезок разбиения стремится к нулю, а точка ξ_i принадлежит i-му отрезку разбиения и обозначается

$$\int_a^b f(x)\,dx = \lim_{n \to \infty} \sum_{i=1}^{n} f(\xi_i)\Delta x_i$$

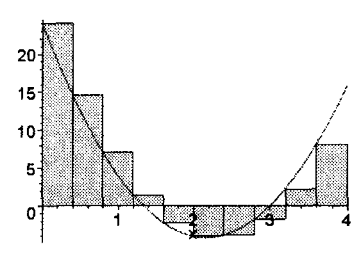
Геометрически определенный интеграл представляет площадь криволинейной трапеции под графиком положительной функции. Если на некотором отрезке функция отрицательна, то определенный интеграл по этому промежутку отрицателен.

В пакете `student` есть команда `leftbox()`, которая строит график функции и отображает интегральную сумму в виде последовательности прямоугольников для заданного разбиения промежутка интегрирования, причем точки, в которых вычисляются значения функции для интегральной суммы, соответствуют левым концам отрезков разбиения (ее называют левой суммой в противоположность правой сумме, в которой значения функции берутся на правых концах отрезков разбиения интервала интегрирования):

```
> f:=x->-(x-2)*(x-3)*(x-4)+x^2*(x-3);
```

$$f := x \to -(x-2)(x-3)(x-4) + x^2 (x-3)$$

```
> with(student):
> leftbox(f(x),x=0..4,10);
```



Команда `leftsum()` вычисляет левую сумму для заданного разбиения, которая аппроксимирует определенный интеграл от заданной функции на заданном промежутке:

```
> leftsum(f(x),x=0..4,10);
```

$$\frac{2}{5}\left(\sum_{i=0}^{9}\left(-\left(\frac{2}{5}i-2\right)\left(\frac{2}{5}i-3\right)\left(\frac{2}{5}i-4\right)+\frac{4}{25}i^2\left(\frac{2}{5}i-3\right)\right)\right)$$

```
> evalf(%);
```

$$18.24000000$$

Увеличивая число отрезков разбиения заданного промежутка, мы все точнее и точнее будем вычислять площадь криволинейной трапеции, а тем самым и определенный интеграл:

```
> boxes:=[seq(i^2,i=2..20)];
```

boxes := [4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400]

```
> seq(evalf(leftsum(f(x),x=0..4,n)), n=boxes);
 24., 18.56790123, 17.25000000, 16.74240000, 16.49382715, 16.35318617,
      16.26562500, 16.20728548, 16.16640000, 16.13660269, 16.11419753,
      16.09691537, 16.08329863, 16.07237531, 16.06347656, 16.05612959,
      16.04999238, 16.04481242, 16.04040000
> int(f(x),x=0..4);
```

$$16$$

Мы видим, что последовательность левых сумм постепенно приближается к точному значению интеграла, которое равно 16.

Можно создать анимационную картинку изменения отображения левых сумм, добавив в качестве заголовка ее значение при соответствующем числе разбиений промежутка интегрирования:

```
> S:=seq(leftbox(f(x),x=0..4,n,
        title=convert(evalf(leftsum(f(x),x=0..4,n)),string)),
        n=boxes):
```

```
> with(plots):
> display(S,insequence=true);
```

24. 16.20728548 16.08329863

Замечание

На рисунке представлены три кадра анимационного изображения: первый, восьмой и тринадцатый.

В пакете `student` имеются команды `middlebox()` и `rightbox()`, аналогичные `leftbox()`, но строящие прямоугольники с высотой, равной значениям функций, соответственно, в средних и правых крайних точках отрезков деления промежутка интегрирования, а также аналогичные команде `leftsum()` команды `middlesum()` и `rightsum()`. Возможности Maple позволяют отобразить графики изменения левой, средней и правой сумм в зависимости от числа отрезков разбиения интервала интегрирования:

```
> SL:=[seq([n,evalf(leftsum(f(x),x=0..4,n))], n=boxes)]:
> SM:=[seq([n,evalf(middlesum(f(x),x=0..4,n))], n=boxes)]:
> SR:=[seq([n,evalf(rightsum(f(x),x=0..4,n))], n=boxes)]:
> plot([SL,SM,SR],view=[0..200,14..20],
         color=black,thickness=2,linestyle=[1,4,7]);
```

На рисунке график левой суммы представлен сплошной линией, средней — штрихпунктирной линией и правой — точечной линией. По горизонтальной оси координат откладывается число отрезков разбиения. Видно, что при небольшом числе отрезков разбиения для нашей функции правая сумма совершает одно колебание, но с увеличением числа отрезков характер изменения всех трех сумм стабилизируется: значения средней суммы лежат между значениями правой и верхней, причем значения правой суммы меньше значений левой.

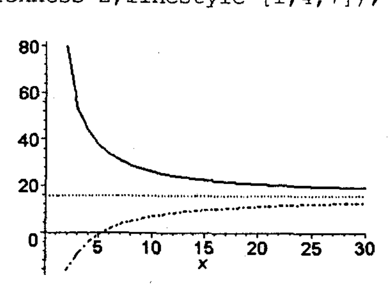
При строгом определении интеграла в рассмотрение вводят нижние и верхние суммы Дарбу, в которых значение функции на отрезках разбиения представляет, соответственно, минимальное и максимальное значение функции на отрезке. Создадим две процедуры, вычисляющие суммы Дарбу подынтегральной функции:

```
> lowDarbou:=proc(f::anything,x::name,s::`..`,n::integer)
                local i,dx,x0;
                dx:=(op(2,s)-op(1,s))/n;
                sum(dx*minimize(f,x=op(1,s)+dx*(i-1)..dx*i),i=1..n);
                end proc:
> highDarbou:=proc(f::anything,x::name,s::`..`,n::integer)
                local i,dx,x0;
                dx:=(op(2,s)-op(1,s))/n;
                sum(dx*maximize(f,x=op(1,s)+dx*(i-1)..dx*i),i=1..n);
                end proc:
```

Параметрами этих процедур являются подынтегральная функция `f`, имя ее независимой переменной `x`, диапазон интегрирования `s` и число промежутков разбиения интервала интегрирования `n`. Для нахождения минимального и максимального значений функции на заданном промежутке использованы команды `minimize()` и `maximize()`.

В курсе математического анализа доказывается, что значение нижней суммы Дарбу всегда меньше значения верхней суммы Дарбу при любом числе промежутков разбиения интервала интегрирования. Это означает, что график верхней суммы Дарбу как функции от числа промежутков разбиения расположен всегда выше графика нижней суммы Дарбу. При стремлении количества промежутков разбиения к бесконечности эти суммы сходятся к одному и тому же числу, которое равняется значению определенного интеграла от заданной функции на заданном интервале. В Maple все эти предложения можно легко проверить графическим способом:

```
> high:=[seq([n,highDarbou(f(x),x,0..4,n)],n=2..30)]:
> low:=[seq([n,lowDarbou(f(x),x,0..4,n)],n=2..30)]:
> plot([high,low,int(f(x),x=0..4)],x=0..30,
       color=black,thickness=2,linestyle=[1,4,7]);
```



На рисунке верхняя сумма Дарбу представлена сплошной, нижняя — штрих-пунктирной, а значение интеграла точечной линиями. Конечно, при таком небольшом разбиении, которое выбрано нами, суммы Дарбу еще значительно отличаются от значения интеграла, но если вычислить их при большом значении n, то мы увидим, что они действительно сходятся к нему, правда достаточно медленно:

```
> int(f(x),x=0..4);
```

$$16$$

```
> evalf(highDarbou(f(x),x,0..4,300));
```

$$16.32292978$$

```
> evalf(highDarbou(f(x),x,0..4,500));
```

$$16.19358925$$

```
> evalf(lowDarbou(f(x),x,0..4,300));
```

$$15.67848889$$

```
> evalf(lowDarbou(f(x),x,0..4,500));
```

$$15.80692267$$

Определенные интегралы широко используются в геометрии, физике и механике. С их помощью легко вычисляются длины плоских кривых и площади криволинейных областей, пройденный точкой путь, произведенная работа и многие другие механические и физические величины.

Задача 9.3

Вычислить длину дуги цепной линии, заданной уравнением $y=a\ ch(x/a)$, от точки $x=0$ до текущей координаты x.

Решение. Известно, что дифференциал длины дуги кривой, заданной явным уравнением $y=f(x)$, выражается через производную следующим образом:

$$\sqrt{1+\left(\frac{\partial}{\partial x}\mathrm{f}(x)\right)^2}\,dx$$

Тогда длина дуги кривой, ограниченной точками $x=x_0$ и $x=x_1$, определяется через интеграл в виде:

$$s=\int_{x_0}^{x_1}\sqrt{1+\left(\frac{\partial}{\partial x}\mathrm{f}(x)\right)^2}\,dx$$

Вычислим для цепной функции производную от длины переменной дуги:

```
> f:=x->a*cosh(t/a);
```

$$f:=x\to a\cosh\left(\frac{t}{a}\right)$$

```
> `s'`:=simplify(sqrt(1+diff(f(t),t)^2));
```

$$s' := \operatorname{csgn}\left(\cosh\left(\frac{t}{a}\right)\right)\cosh\left(\frac{t}{a}\right)$$

В результате Maple использовал функцию вычисления знака `csgn()` для выражения `cosh(t/a)` (мы помним, что гиперболический синус в англоязычной математической литературе обозначается $\sinh(x)$, гиперболический косинус $\cosh(x)$). Дело в том, что параметром в команде упрощения `simplify()` является выражение, содержащее квадратный корень:

```
> sqrt(1+diff(f(t),t)^2);
```

$$\sqrt{1+\sinh\left(\frac{t}{a}\right)^2}$$

Под корнем стоит выражение, которое упрощается до квадрата функции `cosh(t/a)`, а так как результат извлечения квадратного корня из квадрата действительного числа зависит от знака числа, то Maple как раз и использует функцию `csgn()` для отражения этого факта. Однако мы знаем, что функция гиперболического косинуса всегда положительна, а поэтому можем немного упростить полученное выражение, убрав из него функцию вычисления знака:

```
> `s'`:=op(2,`s'`);
```

$$s' := \cosh\left(\frac{t}{a}\right)$$

Теперь, имея выражение s' для дифференциала дуги цепной линии, можно вычислить и ее длину:

```
> Int(`s'`,t=0..x)=int(`s'`,t=0..x);
```

$$\int_0^x \cosh\left(\frac{t}{a}\right)dt = a\sinh\left(\frac{x}{a}\right)$$

Используя аналогичный подход, можно вычислять длины дуг параметрически заданных кривых. Единственное, что надо помнить, это формулу дифференциала длины дуги параметрически заданной кривой (x=x(t), y=y(t)):

$$\sqrt{\left(\frac{\partial}{\partial t}\mathrm{x}(t)\right)^2+\left(\frac{\partial}{\partial t}\mathrm{y}(t)\right)^2}\,dt$$

Задача 9.4

Вычислить длину дуги четверти астроиды $x=a\cos^3 t$, $y=a\sin^3 t$ $(0\le t\le\pi/2)$.

Решение. Начертим сначала график астроиды и выделим дугу, длину которой необходимо вычислить:

```
> x:=a*cos(t)^3; y:=a*sin(t)^3;
```

$$x := a\cos(t)^3$$

$$y := a\sin(t)^3$$

```
> f:=plot([eval(x,a=1),eval(y,a=1),t=0..2*Pi],color=black,
            linestyle=7,thickness=2):
  g:=plot([eval(x,a=1),eval(y,a=1),t=0..Pi/2],color=black,
            linestyle=1,thickness=2):
  with(plots):
  display({f,g},scaling=CONSTRAINED);
```

На рисунке астроида отображается точечной линией, а ее четверть сплошной линией.

Теперь вычислим производную длины дуги астроиды и определенный интеграл от нее на заданном интервале изменения параметра *t*, величина которого и будет равна длине четверти астроиды:

```
> `s'`:=simplify(sqrt(diff(x,t)^2+diff(y,t)^2));
```

$$s' := 3\sqrt{-\cos(t)^2 a^2 (-1+\cos(t)^2)}$$

```
> Int(`s'`,t=0..Pi/2)=int(`s'`,t=0..Pi/2);
```

$$\int_0^{1/2\,\pi} 3\sqrt{-\cos(t)^2 a^2 (-1+\cos(t)^2)}\,dt = \frac{3}{2}a$$

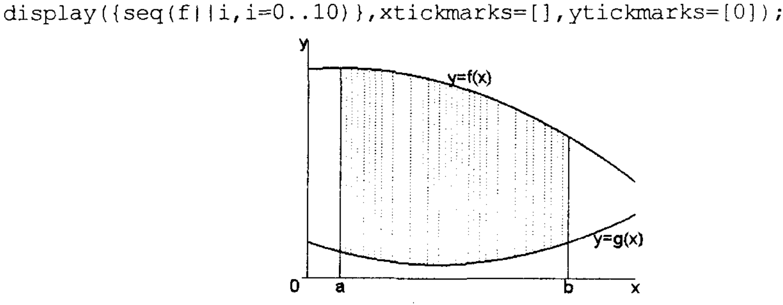
Из определения определенного интеграла нам известно, что если подынтегральная функция f(x) на некотором промежутке [a,b] изменения независимой переменной x положительна, то интеграл равен площади криволинейной трапеции — фигуры, ограниченной сверху кривой y=f(x), слева и справа двумя ординатами x=a и x=b и снизу отрезком [a,b] оси x. Если криволинейная трапеция ограничена снизу не отрезком оси x, а также некоторой кривой y=g(x) (причем эта кривая не обязательно должна быть положительна), то площадь подобной фигуры определяется по формуле:

$$S = \int_a^b f(x) - g(x)\,dx$$

Рисунок примера 9.1 иллюстрирует сказанное выше относительно вычисления площади криволинейной трапеции.

Пример 9.1. Вычисление площади криволинейной трапеции

```
> with(plots):
  g2:=-0.1*(x-1)^2+15:
  g1:=0.1*(x-4)^2+1:
  n:=40:
  f0:=plot(g1,x=0..10,0..17,color=black,thickness=2):
  f1:=plot(g2,x=0..10,color=black,thickness=2):
  f2:=seq(plot([[1+7/(n-1)*(i-1),eval(g2,x=1+7/(n-1)*(i-1))],
    [1+7/(n-1)*(i-1),eval(g1,x=1+7/(n-1)*(i-1))]],color=gray),i=2..n-1):
  f3:=plot([[1,0],[1,eval(g2,x=1)]],color=black):
  f4:=plot([[8,0],[8,eval(g2,x=8)]],color=black):
  f5:=textplot([1,0,"a"],align=BELOW):
  f6:=textplot([8,0,"b"],align=BELOW):
  f7:=textplot([0,17,"y"],align=LEFT):
  f8:=textplot([10,0,"x"],align=BELOW):
  f9:=textplot([5,14,"y=f(x)],align=ABOVE):
  f10:=textplot([9.5,3.5,"y=g(x)],align=BELOW):
  display({seq(f||i,i=0..10)},xtickmarks=[],ytickmarks=[0]);
```



Замечание

Мы сочли уместным привести полный текст "программы" на языке Maple создания рисунка примера 9.1, чтобы читатель мог в дальнейшем достаточно легко создавать подобные "общие" чертежи, которые часто встречаются в математической литературе. Обратите внимание, что задание пустого списка засечек горизонтальной оси в опции `xtickmarks`, а также одного значения `0` для засечек по вертикальной оси приводит к вычерчиванию осей координат вообще без засечек с единственной маркированной точкой — началом координат.

Задача 9.5

Найти площадь фигуры, ограниченной осями координат и параболой

$$\sqrt{x}+\sqrt{y}=\sqrt{a}, \quad (a>0).$$

Решение. Прежде всего выразим уравнение параболы в явном виде и начертим ее график. Из задания параболы ясно, что независимая переменная x должна быть неотрицательной:

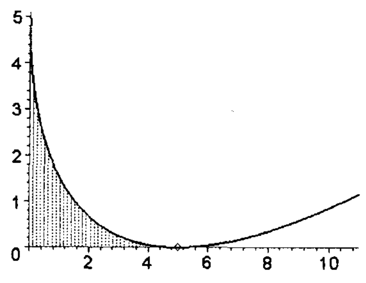
```
> f:=sqrt(x)+sqrt(y)=sqrt(a);
```

$$f := \sqrt{x} + \sqrt{y} = \sqrt{a}$$

```
> f:=rhs(isolate(f,y));
```

$$f := (\sqrt{a} - \sqrt{x})^2$$

```
> with(plots):
  a1:=a:
  n:=40:
  f0:=plot(eval(f,a=5),x=0..11,0..7,color=black,thickness=2):
  f1:=seq(plot([[eval(a1,a=5)/(n-1)*(i-1),0],
              [eval(a1,a=5)/(n-1)*(i-1),
              eval(f,[x=eval(a1,a=5)/(n-1)*(i-1),a=5])]]),i=1..n):
  f2:=plot([[eval(a1,a=5),0]],style=POINT, symbol=DIAMOND, symbolsize=16,
                              color=black):
  f3:=textplot([0.1,5,"a"],align=RIGHT):
  f4:=textplot([5,0,"a"],align=BELOW):
  f5:=textplot([0,7,"y"],align=LEFT):
  f6:=textplot([11,0,"x"],align=BELOW):
  display({f0,f1,f2,f3,f4,f5,f6},xtickmarks=[],ytickmarks=[0]);
```



Точку пересечения графика параболы с осью абсцисс найдем из решения уравнения:

```
> p:=solve(f,x);
```

$$p := a$$

Теперь можно вычислить искомую площадь как интеграл:

```
> Int(f,x=0..p)=int(f,x=0..p);
```

$$\int_0^a (\sqrt{a} - \sqrt{x})^2 \, dx = \frac{1}{6} a^2$$

Задача 9.6

Определить площадь фигуры, заключенной между двумя конгруэнтными параболами $y^2=2px$ и $x^2=2py$.

Решение. Как и в предыдущей задаче, сначала мы построим графики парабол, чтобы выяснить их взаимное расположение:

```
> ff:=y^2=2*p*x;
```

$$ff := y^2 = 2\,p\,x$$

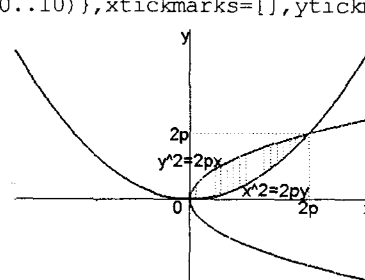
```
> f:=sqrt(2*p*x);
```

$$f := \sqrt{2}\,\sqrt{p\,x}$$

```
> g:=x^2/2/p;
```

$$g := \frac{1}{2}\,\frac{x^2}{p}$$

```
> with(plots):
  n:=20:
  f0:=implicitplot(eval(ff,p=1),x=-3..3,y=-3..3,color=black,thickness=2):
  f1:=plot(eval(g,p=1),x=-3..3,color=black,thickness=2):
  f2:=seq(plot([[2./(n-1)*(i-1),eval(f,[p=1,x=2/(n-1)*(i-1)])],
      [2/(n-1)*(i-1),eval(g,[p=1,x=2/(n-1)*(i-1)])]],color=gray),i=1..n):
  f3:=plot([[2,0],[2,2]],color=black,linestyle=7):
  f4:=plot([[0,2],[2,2]],color=black,linestyle=7):
  f5:=textplot([2,0,"2p"],align=BELOW):
  f6:=textplot([0,2,"2p"],align=LEFT):
  f7:=textplot([0,5,"y"],align=LEFT):
  f8:=textplot([3,0,"x"],align=BELOW):
  f9:=textplot([0,1,"y^2=2px"],align=ABOVE):
  f10:=textplot([2,0.1,"x^2=2py"],align={ABOVE,LEFT}):
  display({seq(f||i,i=0..10)},xtickmarks=[],ytickmarks=[0]);
```



Из рисунка видно, что две параболы пересекаются в двух точках, координаты которых можно определить из решения системы уравнений:

```
> eq1:=y=f;eq2:=y=g;
```

$$eq1 := y = \sqrt{2}\,\sqrt{p\,x}$$

$$eq2 := y = \frac{1}{2}\,\frac{x^2}{p}$$

```
> c:=solve({eq1,eq2},{x,y});
```

$$c := \{y=0, x=0\}, \{y=\sqrt{2}\sqrt{-p^2(\sqrt{2}\,\text{RootOf}(\%1, -.7071067810 + 1.224744871\,I)+2)},$$
$$x=-p(\sqrt{2}\,\text{RootOf}(\%1, -.7071067810 + 1.224744871\,I)+2)\}, \{y=2\,p, x=2\,p\}, \{$$
$$x=-p(\sqrt{2}\,\text{RootOf}(\%1, -.7071067810 - 1.224744871\,I)+2),$$
$$y=\sqrt{2}\sqrt{-p^2(\sqrt{2}\,\text{RootOf}(\%1, -.7071067810 - 1.224744871\,I)+2)}\}$$
$$\%1 := _Z^2 + 2 + \sqrt{2}\,_Z$$

Из полученного множества решений нас, естественно, интересуют только действительные $\{x=0,\ y=0\}$ и $\{x=2p,\ y=2p\}$, из которых определяется интервал интегрирования для определения площади фигуры, образованной пересечением конгруэнтных парабол, — $[0,\ 2p]$. Определенный интеграл от разности функций `f` и `g` по этому промежутку и будет равен площади искомой фигуры:

```
> S:=int(f-g,x=0..2*p);
```

$$S := \frac{8}{3}p\sqrt{p^2} - \frac{4}{3}p^2$$

```
> subs(sqrt(p^2)=p,S);
```

$$\frac{4}{3}p^2$$

Если параметр $p > 0$, то можно упростить выражение для площади, заменив в нем $\sqrt{p^2}$ на p.

Кроме вычисления длин дуг кривых и площадей определенный интеграл применяется при вычислении объемов тел вращения, образуемых вращением криволинейной трапеции, ограниченной сверху неотрицательной функцией $y=\mathrm{f}(x)$, слева и справа прямыми $x=a$ и $x=b$ ($a \le b$), а снизу отрезком $[a,\ b]$ оси x, вокруг этой оси. В этом случае объем полученного тела можно вычислить по формуле

$$V = \pi\int_a^b \mathrm{f}(x)^2\,dx.$$

Если криволинейная трапеция ограничена и сверху и снизу кривыми $y=\mathrm{f}(x)$ и $y=\mathrm{g}(x)$, то, очевидно,

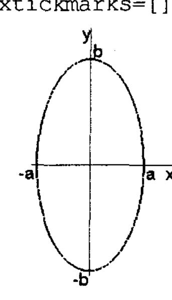
$$V = \pi\int_a^b \mathrm{f}(x)^2 - \mathrm{g}(x)^2\,dx.$$

Задача 9.7

Вычислить объем эллипсоидов вращения, образованных вращением эллипса $\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$ вокруг оси x и оси y.

Решение. Нарисуем на плоскости *xy* заданный в задаче эллипс:

```
> with(plots):
  f0:=implicitplot(x^2/2^2+y^2/4^2=1,x=-2..2,y=-4..4,
              color=black,thickness=2,scaling=CONSTRAINED):
  f1:=textplot([2.1,0,"a"],align={RIGHT,BELOW}):
  f2:=textplot([-2.1,0,"-a"],align={LEFT,BELOW}):
  f3:=textplot([0,5,"y"],align=LEFT):
  f4:=textplot([3,0,"x"],align=BELOW):
  f5:=textplot([0.1,4.1,"b"],align={ABOVE,RIGHT}):
  f6:=textplot([0,-4,"-b"],align={BELOW,LEFT}):
  display({seq(f||i,i=0..6)},xtickmarks=[],ytickmarks=[]);
```



Для вычисления объема эллипсоида вращения, образованного вращением вокруг оси x, следует найти квадрат координаты y, а потом вычислить в соответствии с приведенной выше формулой определенный интеграл на промежутке $[-a, a]$:

```
> ellips:=x^2/a^2+y^2/b^2=1;
```

$$ellips := \frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

```
> isolate(ellips,y^2);
```

$$y^2 = \left(1 - \frac{x^2}{a^2}\right) b^2$$

```
> f:=rhs(%);
```

$$f := \left(1 - \frac{x^2}{a^2}\right) b^2$$

```
> V[x]=Pi*Int(f,x=-a..a);
```

$$V_x = \pi \int_{-a}^{a} \left(1 - \frac{x^2}{a^2}\right) b^2 \, dx$$

```
> V[x]=Pi*int(f,x=-a..a);
```

$$V_x = \frac{4}{3} \pi \, a \, b^2$$

При вращении эллипса вокруг оси y нам необходимо вычислить определенный интеграл от квадрата координаты x на промежутке $[-b, b]$:

```
> isolate(ellips,x^2);
```

$$x^2 = \left(1 - \frac{y^2}{b^2}\right) a^2$$

```
> f:=rhs(%);
```

$$f := \left(1 - \frac{y^2}{b^2}\right) a^2$$

```
> V[y]=Pi*Int(f,y=-b..b);
```

$$V_y = \pi \int_{-b}^{b} \left(1 - \frac{y^2}{b^2}\right) a^2 \, dy$$

```
> V[y]=Pi*int(f,y=-b..b);
```

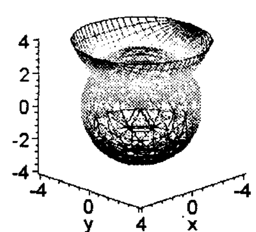
$$V_y = \frac{4}{3} \pi \, b \, a^2$$

Задача 9.8

Найти объем общей части параболоида вращения $2az=x^2+y^2$ и сферы $x^2+y^2+z^2=3a^2$.

Решение. Оба тела являются телами вращения относительно оси z, а поэтому и общая их часть также будет телом вращения относительно этой же оси:

```
> g:=plot3d((x^2+y^2)/2/2,x=-4..4,y=-sqrt(16-x^2)..sqrt(16-x^2)):
  g1:=implicitplot3d(x^2+y^2+z^2=12,x=-4..4,y=-4..4,z=-4..4):
  display({g,g1},scaling=CONSTRAINED,axes=FRAME,
                    shading=ZHUE,style=WIREFRAME,orientation=[45,70]);
```



Замечание

Для иллюстрации пересечения параболоида вращения и сферы мы задали значение параметра *a* равным 2.

Для вычисления объема тела, получаемого пересечением параболоида вращения и сферы, необходимо определить уравнение кривой, вращение кото-

рой образует их общую часть. Для этого следует построить сечения заданных тел плоскостью, проходящей через ось z, например, xz:

```
> p:=2*a*z=x^2+y^2;
```

$$p := 2\,a\,z = x^2 + y^2$$

```
> s:=x^2+y^2+z^2=3*a^2;
```

$$s := x^2 + y^2 + z^2 = 3\,a^2$$

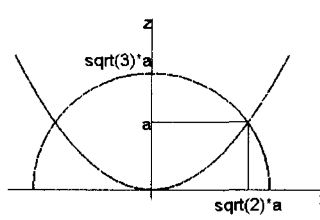
```
> p[z]:=subs(y=0,p);
```

$$p_z := 2\,a\,z = x^2$$

```
> s[z]:=subs(y=0,s);
```

$$s_z := x^2 + z^2 = 3\,a^2$$

```
> f0:=implicitplot({eval(p[z],a=2),eval(s[z],a=2)},x=-4..4,z=0..12,
                      color=black,thickness=2,scaling=CONSTRAINED):
  f1:=plot([[0,2],[sqrt(8),2]],color=black):
  f2:=textplot([0,2,"a"],align={LEFT}):
  f3:=textplot([0,sqrt(12)+0.2,"sqrt(3)*a"],align={LEFT,ABOVE}):
  f4:=textplot([0,5,"z"],align=LEFT):
  f5:=textplot([5,0,"x"],align=BELOW):
  f6:=plot([[sqrt(8),0],[sqrt(8),2]],color=black):
  f7:=textplot([sqrt(8),-0.2,"sqrt(2)*a"],align=BELOW):
  display({seq(f||i,i=0..7)},xtickmarks=[],ytickmarks=[]);
```



Точки пересечения сечений заданных в задаче тел вращения можно вычислить следующими командами:

```
> subs(x^2=lhs(p[z]),s[z]);
```

$$2\,a\,z + z^2 = 3\,a^2$$

```
> solve(%,z);
```

$$a, -3\,a$$

```
> z[1]:=%[1];
```

$$z_1 := a$$

```
> x[1]:=simplify(subs(z=z[1],sqrt(lhs(p[z]))),symbolic);
```

$$x_1 := \sqrt{2}\,a$$

Так как параболоид вращения задан только при $z \geq 0$, то из двух значений z выбирается первое, которое больше нуля.

Теперь ясно, что сечение тела вращения, являющегося пересечением параболоида и сферы, представляется параболой $2az=x^2$, пока z принадлежит промежутку $[0, a]$, и дугой окружности $x^2+z^2=3a^2$, когда z изменяется в интервале $[a, \sqrt{3}\,a]$. Поэтому объем тела, полученного вращением составной кривой вокруг оси z, представляется суммой двух интегралов:

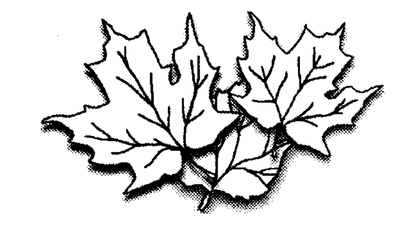
```
> V=Pi*Int(lhs(p[z]),z=0..a)+
            Pi*Int(rhs(isolate(s[z],x^2)),z=a..sqrt(3)*a);
```

$$V = \pi \int_0^a 2\,a\,z\,dz + \pi \int_a^{\sqrt{3}\,a} 3\,a^2 - z^2\,dz$$

```
> V=simplify(Pi*int(lhs(p[z]),z=0..a)+
            Pi*int(rhs(isolate(s[z],x^2)),z=a..sqrt(3)*a));
```

$$V = -\frac{5}{3}\pi\,a^3 + 2\,\pi\sqrt{3}\,a^3$$

ГЛАВА 10



Ряды и дифференциальные уравнения

Maple 6 предоставляет пользователю несколько инструментов для решения обыкновенных дифференциальных уравнений: команду `dsolve()` *(см. гл. 2)* и пакет `DEtools` *(см. гл. 3)*. Первая пытается найти общее решение дифференциального уравнения в замкнутой форме. Однако это не всегда возможно, поэтому в состав Maple включен специальный пакет `DEtools` построения численного решения задачи Коши и его графика. Часто случается так, что численное решение не совсем устраивает пользователя по тем или иным соображениям, и он хочет получить приближенное аналитическое решение поставленной задачи Коши. В этом случае можно воспользоваться классическим методом построения приближенного решения разложением его в подходящий ряд, например ряд Тейлора, в окрестности начальной точки. В этой главе мы покажем, как можно эффективно использовать Maple 6 для построения приближенных аналитических решений дифференциальных уравнений, а также покажем возможности Maple при работе со специальными функциями: функциями Дирака и Хевисайда.

10.1. Дифференциальные уравнения с разрывными правыми частями

Многие реальные явления можно с определенной степенью точности описать дифференциальными уравнениями, которые представляют его математическую модель. При их выводе прибегают к различным упрощениям, вводя так называемые гипотезы. Например, не существует в природе сосредоточенной силы, но при построении математических моделей, или схем расчета балок ее с успехом применяют для математического моделирования силы, действующей по очень маленькой площадке нагружения.

Для математического описания сосредоточенной силы используется обобщенная δ-функция Дирака, которая равна нулю всюду за исключением ну-

ля, где она имеет сингулярность (ее значение равно бесконечности), а определенный интеграл от нее по всей действительной оси равен 1:

```
> int(Dirac(x),x=-infinity..infinity);
```

$$1$$

Функция Дирака в механике рассматривается как единичная сосредоточенная сила. В Maple для задания δ-функции Дирака используется команда `Dirac(x)`, производная этой функции представляется функцией `Dirac(n,x)`, где `n` является порядком вычисляемой производной. Если попытаться вычислить производную δ-функции обычной командой `diif()`, то результат будет представлен указанной выше функцией:

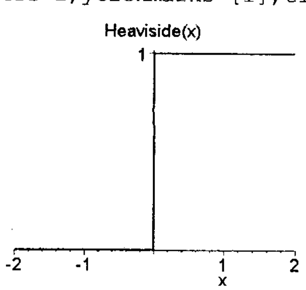
```
> diff(Dirac(x),x$4);
```

$$\mathrm{Dirac}(4, x)$$

Если необходимо определить сингулярность, а тем самым и сосредоточенную силу, в точке, отличной от нуля, то следует использовать преобразование смещения. Например, `Dirac(x-1/2)` определяет функцию Дирака с сингулярностью в точке `x=1/2`.

Для задания распределенных нагрузок используется другая обобщенная функция — функция Хевисайда `Heaviside(x)`, которая равна `0` при `x<0` и `1` при `x>0`, в точке `x=0` эта функция не определена. Таким образом, функция Хевисайда имеет единичный скачок в точке `x=0`:

```
> plot(Heaviside(x),x=-2..2,
       color=black,thickness=2,ytickmarks=[1],title="Heaviside(x));
```



Производная функции Хевисайда равняется δ-функции Дирака:

```
> diff(Heaviside(x),x);
```

$$\mathrm{Dirac}(x)$$

Распределенную нагрузку с интенсивностью q, действующую на балку, начиная, например, с точки $x=l/2$, где l — длина балки, можно задать с помощью функции Хевисайда в виде:

$$Q = q\ \mathrm{Heaviside}\left(x - \frac{1}{2}\,l\right)$$

Если распределенная нагрузка действует на интервале $[a, b]$ по длине балки (значения a и b меньше длины l), то такая нагрузка представляется разностью функций Хевисайда:

$$Q = q\,(\text{Heaviside}\,(x - a) - \text{Heaviside}\,(x - b))$$

Maple может решать дифференциальные уравнения, в которых коэффициенты определены через две рассмотренные обобщенные функции. Рассмотрим следующее дифференциальное уравнение:

```
> del:=diff(EJ*diff(y(x),x$2),x$2)=Dirac(x-1/2)+Heaviside(x-1);
```

$$del := EJ\left(\frac{\partial^4}{\partial x^4}\,y(x)\right) = \text{Dirac}\left(x - \frac{1}{2}\right) + \text{Heaviside}\,(x - 1)$$

Оно представляет уравнение изгиба балки постоянной жесткости EJ, нагруженной единичной сосредоточенной силой в точке, отстоящей на 1/2 от левого конца балки, определенного в точке начала координат, и распределенной нагрузкой единичной интенсивности, начинающейся на расстоянии 1 от левого конца балки и действующей до правого конца балки.

Для решения уравнения изгиба балки следует задать граничные условия на ее концах:

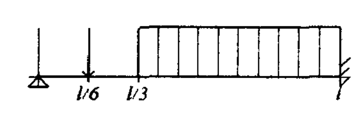
```
> boundary={y(0)=0,(D@@2)(y)(0)=0,y(3)=0,(D@@1)(y)(3)=0};
```

$$boundary = \{\,y(0) = 0, (D^{(2)})(y)(0) = 0, y(3) = 0, D(y)(3) = 0\,\}$$

На левом конце балки функция прогиба $y(x)$ и ее вторая производная равны нулю, на правом конце (x=3) функция прогиба и ее первая производная равны нулю. С точки зрения механики эти условия соответствуют свободному опиранию левого конца балки и жесткому защемлению ее правого конца. Ниже представлена расчетная схема балки, которая построена с помощью пакета `geometry`:

```
> with(geometry):
> # Построение линии балки
> point(p1,[0,0]):point(p2,[3,0]):segment(s1,[p1,p2]):
> # Построение опоры левого конца балки
> point(p0,[0.1,-0.1]):point(p3,[-0.1,-0.1]):triangle(t1,[p0,p3,p1]):
> # Построение заделки правого конца балки
> point(p4,[3,-0.1]):point(p5,[3,0.1]):segment(s2,[p4,p5]):
> point(p6,[3,-0.1]):point(p7,[3.1,0]):segment(s3,[p6,p7]):
> point(p8,[3,0]):point(p9,[3.1,0.1]):segment(s4,[p8,p9]):
> point(p10,[3,0.1]):point(p11,[3.1,0.2]):segment(s5,[p10,p11]):
> # Построение сосредоточенной нагрузки
> point(p12,[1/2,0]):point(p13,[1/2,0.5]):dsegment(s6,[p12,p13]):
> point(p18,[0.55,0.05]):point(p19,[0.5,0]):dsegment(s10,[p18,p19]):
> point(p20,[0.45,0.05]):dsegment(s11,[p20,p19]):
```

```
> # Построение распределенной нагрузки
> point(p14,[1,0]):point(p15,[1,0.5]):dsegment(s7,[p14,p15]):
> point(p16,[3,0]):point(p17,[3,0.5]):dsegment(s8,[p16,p17]):
> dsegment(s9,[p17,p15]):
> # Построение штриховки распределенной нагрузки
> d:=2/10:for i from 1 to 10 do
    point(pp||(2*i-2),[1+d*(i-1),0]):point(pp||(2*i-1),[1+d*(i-1),0.5]):
    dsegment(s||(11+i),[pp||(2*i-2),pp||(2*i-1)]):
  end do:
> # Вычерчивание схемы балки
> draw([seq(s||i,i=1..11),seq(s||i,i=12..21)(thickness=1),t1,p1],
              thickness=2,xtickmarks=[0.5="1/6",1="1/3",3="1"],
              ytickmarks=[],axesfont=[TIMES,ITALIC,11],
              color=black,symbol=CIRCLE,axes=NORMAL);
```



Решим поставленную граничную задачу командой `dsolve()`:

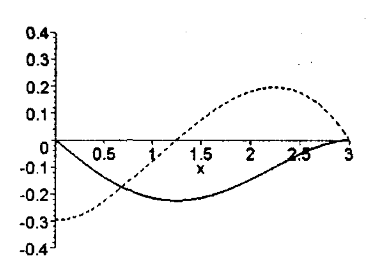
```
> collect(dsolve({de1,y(0)=0,(D@@2)(y)(0)=0,y(3)=0,(D@@1)(y)(3)=0},
                  y(x)),x);
```

$$y(x)=\frac{1}{24}\frac{\text{Heaviside}(x-1)\,x^4}{EJ}+\left(\frac{\frac{1}{6}\%1-\frac{1}{6}\text{Heaviside}(x-1)}{EJ}-\frac{485}{2592}\frac{1}{EJ}\right)x^3$$
$$+\frac{\left(-\frac{1}{4}\%1+\frac{1}{4}\text{Heaviside}(x-1)\right)x^2}{EJ}+\left(\frac{\frac{1}{8}\%1-\frac{1}{6}\text{Heaviside}(x-1)}{EJ}+\frac{\frac{19}{32}}{EJ}\right)x$$
$$+\frac{-\frac{1}{48}\%1+\frac{1}{24}\text{Heaviside}(x-1)}{EJ}$$

$$\%1:=\text{Heaviside}\left(x-\frac{1}{2}\right)$$

Выделим правую часть полученного решения и построим график функции прогиба и ее первой производной, подставив в качестве жесткости значение EJ=2:

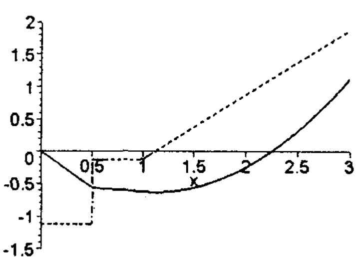
```
> f:=rhs(%):
> plot([eval(-f,EJ=2),eval(-diff(f,x),EJ=2)],x=0..3,-0.4..0.4,
                      color=black,linestyle=[1,4],
                      thickness=2,xtickmarks=[0,1/2,1,3/2,2,2.5,3]);
```



Сплошная линия графика соответствует функции прогиба балки, а пунктирная — углу поворота сечения, перпендикулярного нейтральной оси балки.

Вторая и третья производные функции прогиба, умноженные на *EJ*, равны, соответственно, изгибающему моменту и перерезывающей силе в сечениях балки:

```
> plot([eval(EJ*diff(f,x$2),EJ=2),eval(EJ*diff(f,x$3),EJ=2)],
                         x=0..3,-1.5..2,color=black,linestyle=[1,4],
                         thickness=2,xtickmarks=[0,1/2,1,3/2,2,2.5,3]);
```



Здесь сплошная линия представляет изгибающий момент, а пунктирная — перерезывающую силу.

График прогиба балки, а также построенных на его основе графиков угла поворота, изгибающего момента и перерезывающей силы по длине балки полностью согласуется со схемой ее нагружения: в точке приложения сосредоточенной силы график перерезывающей силы имеет скачок на величину сосредоточенной силы, а для кривой изгибающих моментов в этой точке наблюдается слом.

Как видим, Maple умеет решать (и даже в аналитическом виде), в общем-то, достаточно сложное дифференциальное уравнение, в котором правая часть задана разрывными функциями, причем решение построено также в терминах разрывных функций. Однако Maple не всегда может решить дифференциальное уравнение в замкнутом аналитическом виде. В этом случае можно прибегнуть к построению численного или приближенного аналитического решения. Классический метод построения приближенного решения дифференциального уравнения — построение его в форме степенного ряда Тейлора или Маклорена. Но прежде чем демонстрировать подобную технику, ко-

ротко остановимся на возможностях Maple, связанных с выполнением операций над бесконечными рядами.

10.2. Функциональные ряды

Maple предоставляет несколько команд для разложения функций и выражений в ряды. Одной из них является команда series() со следующим синтаксисом:

```
series(выражение, x=a [, n]);
```

Эта команда строит "обобщенный" степенной ряд для выражения или функции, заданных первым параметром, в окрестности точки a независимой переменной x. Если второй параметр вычисляется равным не уравнению, а просто имени переменной, то по умолчанию ищется разложение выражения в окрестности точки 0 изменения независимой переменной. Третий необязательный параметр должен быть целым положительным числом, определяющий порядок удерживаемых в разложении функции членов.

Замечание

Если параметр n не задан, то используется значение системной переменной Order, которое по умолчанию равно 6. Задание порядка разложения в команде series() не влияет на установленное значение переменной Order.

Результатом выполнения команды series() может быть построение ряда Тейлора или Лорана, асимптотического ряда (если a равно бесконечности) или некоторого более общего ряда. Основным критерием для построения "обобщенного ряда" является выполнение следующих ограничений для коэффициентов ряда

$$k1\,(x-a)^{eps} < |coeff_i|$$

$$|coeff_i| < \frac{k2}{(x-a)^{eps}}$$

при любом $eps > 0$, когда независимая переменная x стремится к a. В неравенствах $k1$ и $k2$ являются некоторыми константами. Выполнение указанных ограничений означает, что коэффициенты ряда могут быть функциями независимой переменной при условии, что эти функции растут не быстрее полинома в окрестности точки разложения. Разложение в ряд функций с помощью команды series() демонстрируется в примере 10.1.

Пример 10.1. Разложение функций в обобщенные степенные ряды

```
> series(ln(x),x=1);
```

$$x-1-\frac{1}{2}(x-1)^2+\frac{1}{3}(x-1)^3-\frac{1}{4}(x-1)^4+\frac{1}{5}(x-1)^5+O((x-1)^6)$$

```
> series((x^2+x+1)/(x+1)^2,x=infinity,8);
```

$$1-\frac{1}{x}+\frac{2}{x^2}-\frac{3}{x^3}+\frac{4}{x^4}-\frac{5}{x^5}+O\left(\frac{1}{x^6}\right)$$

```
> series(exp(x^2),x,10);
```

$$1+x^2+\frac{1}{2}x^4+\frac{1}{6}x^6+\frac{1}{24}x^8+O(x^{10})$$

```
> s:=series(x^x,x);
```

$$1+\ln(x)\,x+\frac{1}{2}\ln(x)^2x^2+\frac{1}{6}\ln(x)^3x^3+\frac{1}{24}\ln(x)^4x^4+\frac{1}{120}\ln(x)^5x^5+O(x^6)$$

```
> whattype(s);
```

$$series$$

Обычно результат разложения функции в обобщенный степенной ряд представляется в Maple в форме специальной структуры данных `series`, что видно из последнего разложения примера 10.1.

Нулевой операнд этой структуры данных представляет аргумент разложения `(x-a)`, где `a` является точкой, в окрестности которой построено разложение в обобщенный степенной ряд. Нечетный операнд `op(2*i-1,series)` содержит i-й коэффициент ряда, а в четном `op(2*i,series)` хранится соответствующий целый показатель степени. Последняя пара операндов типа данных `series` содержит символ `O(1)` и целое число, определяющее порядок усечения ряда и соответствующее значению системной переменной `Order` или третьему параметру команды `series()`. Следует отметить, что нулевые коэффициенты ряда не хранятся в наборе операндов типа `series`. Пример 10.2 демонстрирует вычисление операндов ряда `s`.

Пример 10.2. Операнды типа данных series

```
> s:=series(sin(x)^x,x,4);
```

$$s:=1+\ln(x)\,x+\frac{1}{2}\ln(x)^2x^2+\left(-\frac{1}{6}+\frac{1}{6}\ln(x)^3\right)x^3+O(x^4)$$

```
> op(0,s);
```

$$x$$

```
> for i from 1 to nops([op(s)])/2 do
    op(2*i-1,s),op(2*i,s);
  end do;
```

$$1, 0$$

$$\ln(x), 1$$

$$\frac{1}{2}\ln(x)^2, 2$$

$$-\frac{1}{6}+\frac{1}{6}\ln(x)^3, 3$$

$$O(1), 4$$

Кроме многофункциональной команды разложения в обобщенные степенные ряды series() Maple предлагает команды разложения функций в конкретные степенные ряды (табл. 10.1).

Таблица 10.1. *Команды разложения в ряды*

Команда	Описание
taylor(выражение, x=a [,n])	Разложение в ряд Тейлора функции одной переменной в окрестности точки *x=a* по степеням одночлена (*x-a*). Если второй параметр задан в виде одного имени, то разложение в окрестности точки *x*=0. Необязательный параметр n определяет максимальный порядок вычисления членов ряда
poisson(выражение, v [,n [,w]])	Разложение в ряд Тейлора функции нескольких переменных. Параметр v задает в виде списка или множества независимые переменные функции и значения координат точки, в окрестности которой осуществляется разложение, в форме уравнений (см. taylor()). Необязательный параметр n определяет максимальный общий порядок (сумма степеней всех переменных) вычисления членов ряда. Необязательный параметр w задает в форме списка или множества весовые коэффициенты переменных. Например, если вес переменной *x* будет равен 2, то в ряде будут присутствовать ее степени не выше entier(n/2) (entier(x) — целая часть числа x)
mtaylor(выражение, v [,n [,w]])	Разложение в ряд Тейлора функции нескольких переменных. Аналогична команде mtaylor() за исключением того, что если в коэффициентах встречаются произведения функций sin() и cos(), то они представляются в виде суммы тригонометрических функций по кратным аргументам
chebyshev(выражение,x=a..b [,eps])	Разложение функции одной переменной в ряд по полиномам Чебышева на интервале [a, b]. Необязательный параметр eps задает точность вычислений, по умолчанию равен 10^(-Digits). Расположена в пакете numapprox

Таблица 10.1 (окончание)

Команда	Описание
`asympt(выражение,x=a [,n])`	Разложение функции одной переменной в ряд по степеням `1/x`, когда `x` стремится к бесконечности. Получается подстановкой в ряд Маклорена вместо `x` значения `1/x`. Полученный ряд имеет тип суммы '+', а не `series`

Внимание!

Команда `chebyshev()` разложения в ряд по полиномам Чебышева расположена в пакете `numapprox`, остальные команды разложения в ряды находятся в основной библиотеке Maple.

Использование команд разложения в ряды демонстрируется в примере 10.3.

Пример 10.3. Разложение в степенные ряды различными командами

```
> g:=x/(1-x);
```

$$g := \frac{x}{1-x}$$

```
> taylor(g,x,5);
```

$$x + x^2 + x^3 + x^4 + O(x^5)$$

```
> asympt(g,x,5);
```

$$-1 - \frac{1}{x} - \frac{1}{x^2} - \frac{1}{x^3} - \frac{1}{x^4} + O\left(\frac{1}{x^5}\right)$$

```
> `numapprox/chebyshev`(g, x=-1/2..1/2, 0.3);
```

$$.1547005384\ T(0, 2x) + .6188021533\ T(1, 2x) + .1658075373\ T(2, 2x)$$

```
> f := sin(3*w+x)*cos(2*w-y);
```

$$f := \sin(3w + x)\cos(-2w + y)$$

```
> poisson(f,[x,y],3,[2,2]);
```

$$\frac{1}{2}\sin(5w) + \frac{1}{2}\sin(w) + \left(\frac{1}{2}\cos(w) + \frac{1}{2}\cos(5w)\right)x + \left(\frac{1}{2}\cos(w) - \frac{1}{2}\cos(5w)\right)y$$

```
> mtaylor(f,[x,y],3);
```

$$\sin(3w)\cos(2w) + \sin(3w)\sin(2w)y + \cos(3w)x\cos(2w) - \frac{1}{2}\sin(3w)\cos(2w)y^2$$
$$-\frac{1}{2}\sin(3w)x^2\cos(2w) + \cos(3w)x\sin(2w)y$$

Для работы с формальными степенными рядами в Maple предназначен пакет `powseries`. Ряды, с которыми манипулируют команды этого пакета, представляются в форме процедур, создаваемых командой

```
powcreate(последовательность_уравнений)
```

Последовательность уравнений, передаваемых этой команде, задает коэффициенты создаваемого степенного ряда. Первое уравнение вида `имя_ряда(n)=выражение` определяет общий член ряда, а также имя, по которому к нему можно ссылаться. Правая часть этого уравнения представляет выражение, зависящее от переменной `n`, или рекуррентное выражение, зависящее от одного или нескольких значений предыдущих коэффициентов. В этом случае последующие уравнения определяют начальные значения для заданного рекуррентного соотношения (пример 10.4).

Пример 10.4. Создание формальных степенных рядов

```
> with(powseries):
> powcreate(s(n)=(-1)^n/(3*n+1));
> eval(s);
```

proc (*powparm*) ... **end proc**

```
> tpsform(s,y,10);
```

$$1-\frac{1}{2}y+\frac{1}{3}y^2-\frac{1}{4}y^3+\frac{1}{5}y^4-\frac{1}{6}y^5+\frac{1}{7}y^6-\frac{1}{8}y^7+\frac{1}{9}y^8-\frac{1}{10}y^9+O(y^{10})$$

```
> powcreate(p(n)=p(n-1)+p(n-2), p(0)=1, p(1)=1);
> tpsform(p,x);
```

$$1+x+2x^2+3x^3+5x^4+8x^5+O(x^6)$$

В примере 10.4 для представления формального степенного ряда в форме структуры данных `series` использована команда `tpsform()` пакета `powseries`, первым обязательным параметром которой является формальный степенной ряд, вторым неизвестная переменная, относительно которой строится ряд, а третий необязательный параметр задает порядок усечения создаваемого ряда. Если он не задан, то для определения порядка усечения ряда используется значение системной переменной `Order`.

Все значения процедуры, представляющей формальный степенной ряд, хранятся в ее таблице значений, а выражение `имя_ряда(_k)` возвращает общий член ряда:

```
> op(4,op(p));
```

$$\text{table}([0=1, 1=1, _k=\text{p}(_k-1)+\text{p}(_k-2)])$$

```
> p(_k);
```

$$\text{p}(_k-1)+\text{p}(_k-2)$$

Команда `evalpow()` вычисляет выражения, составленные из формальных степенных рядов:

```
> ps:=evalpow(p*s-1/s+sqrt(p)):
> tpsform(ps,x);
```

$$1+\frac{1}{2}x+\frac{67}{24}x^2+\frac{149}{48}x^3+\frac{32999}{5760}x^4+\frac{31841}{3840}x^5+O(x^6)$$

В выражении, вычисляемом командой `evalpow()`, можно использовать арифметические операции +, -, *, / и ^, математические функции `sqrt()`, `exp()`, `sin()`, `sinh()` и т. д., а также специальные команды формирования формальных степенных рядов, представленные в табл. 10.2.

Таблица 10.2. *Команды формирования рядов*

Команда	Описание
`inverse(p)`	Возвращает ряд, обратный ряду `p` относительно операции умножения, и эквивалентный `evalpow(1/p)`
`negative(p)`	Возвращает ряд, все коэффициенты которого противоположны коэффициентам ряда `p`, и эквивалентный `evalpow(-p)`. Коэффициент `p(0) <> 0`
`reversion(p [,s])`	Вычисляет ряд, обратный ряду `p` по отношению к ряду `s`, т. е. произведение полученного ряда на ряд `p` должно равняться ряду `s`. Если ряд `s` не задан, то по умолчанию используется ряд с единственным ненулевым коэффициентом `s(1)=1`. Коэффициенты `p(0)` и `s(0)` должны равняться `0`, а коэффициент `s(1)=1` для корректного построения обратного ряда
`quotient(p,s)`	Вычисляет ряд, представляющий частное от деления ряда `p` на ряд `s`. Ее результат эквивалентен выполнению команды `evalpow(p/s)` или `multiply(s,inverse(p))`
`subtract(p,s)`	Возвращает ряд, представляющий разность рядов `p` и `s`, вычисляемый как ряд, коэффициенты которого получаются вычитанием соответствующих коэффициентов рядов `p` и `s`. Ее результат эквивалентен выполнению команды `evalpow(p-s)` или `powadd(p,negative(s))`
`powadd(p,...,s)`	Вычисляет ряд, представляющий сумму произвольного числа рядов, заданных в качестве параметров команды
`multiply(p,s)`	Вычисляет ряд, представляющий произведение ряда `p` на ряд `s`. Ее результат эквивалентен выполнению команды `evalpow(p*s)`
`powdiff(p)`	Возвращает ряд, представляющий формальную производную ряда `p`, которая строится почленным дифференцированием исходного ряда
`powint(p)`	Возвращает ряд, представляющий формальный интеграл от ряда `p`, который строится почленным интегрированием исходного ряда

Таблица 10.2 (окончание)

Команда	Описание
powexp(p)	Возвращает ряд, представляющий число e в степени ряд p, и эквивалентный exp(p)
powlog(p)	Вычисляет ряд, равный натуральному логарифму от ряда p, и эквивалентный evalpow(log(p))
powsqrt(p)	Вычисляет ряд, равный квадратному корню из ряда p, и эквивалентный evalpow(sqrt(p))
powsin(p) powcos(p) powsec(p) powcsc(p) powtan(p) powcot(p)	Команды, вычисляющие тригонометрические функции формального ряда p, и эквивалентные вычислению выражения командой evalpow() с соответствующей тригонометрической функцией, например, evalpow(sin(p))
powsinh(p) powcosh(p) powsech(p) powcsch(p) powtanh(p) powcoth(p)	Команды, вычисляющие гиперболические тригонометрические функции формального ряда p, и эквивалентные вычислению выражения командой evalpow() с соответствующей тригонометрической функцией, например, evalpow(sinh(p))

Пакет powseries для работы с формальными степенными рядами можно использовать для получения рядов Маклорена различных математических функций. Для примера рассмотрим задачу построения указанного ряда для функции $\arcsin(x)$.

Задача 10.1

Построить ряд Маклорена функции $\arcsin(x)$.

Решение. Известна формула представления этой функции через интеграл с переменным верхним пределом

$$\arcsin(x) = \int_0^x \frac{1}{\sqrt{1-t^2}}\,dt.$$

Создадим формальный ряд разложения функции t^2 и на его основе построим разложение в ряд Маклорена подынтегральной функции:

```
> with(powseries):
> powcreate(t(n)=0,t(2)=1);
> tpsform(t,x);
```

$$x^2$$

```
> tImt:=evalpow(1/sqrt(1-t)):
> tpsform(tInt,t);
```

$$1+\frac{1}{2}t^2+\frac{3}{8}t^4+O(t^6)$$

Теперь для получения разложения функции arcsin(x) остается только вычислить интеграл от ряда `tInt`:

```
> Intt:=powint(tInt):
> tpsform(Intt,x,14);
```

$$x+\frac{1}{6}x^3+\frac{3}{40}x^5+\frac{5}{112}x^7+\frac{35}{1152}x^9+\frac{63}{2816}x^{11}+\frac{231}{13312}x^{13}+O(x^{14})$$

Этот результат полностью согласуется с разложением функции arcsin(x) в ряд Маклорена командой `taylor()`:

```
> taylor(arcsin(x),x,14);
```

$$x+\frac{1}{6}x^3+\frac{3}{40}x^5+\frac{5}{112}x^7+\frac{35}{1152}x^9+\frac{63}{2816}x^{11}+\frac{231}{13312}x^{13}+O(x^{14})$$

Замечание

Использование команд `series()`, `taylor()`, а также команд пакета `powseries` не позволяет получить выражение для общего члена ряда в разложении функций.

10.3. Приближенное решение дифференциальных уравнений

Если невозможно получить решение дифференциального уравнения в замкнутой форме, то в таких случаях прибегают к приближенным методам интегрирования дифференциальных уравнений. Одним из наиболее часто используемых методов является представление решения задачи Коши в окрестности точки x=x_0, в которой заданы начальные условия для частного решения дифференциального уравнения.

Покажем применение одного из вариантов использования рядов для построения решения дифференциального уравнения на примере нелинейного уравнения второго порядка

$$y''=F(x,y,y'),$$

удовлетворяющего начальным условиям

$$y(x_0)=y_0, y'(x_0)=y'_0$$

Предположим, что решение поставленной задачи Коши существует. Будем искать его в форме ряда Тейлора функции решения y(x):

$$y(x) = y(x_0) + (x - x_0)\, y'(x_0) + \frac{1}{2}\, y''(x_0)\,(x - x_0)^2 + \ldots$$

Для построения решения нам необходимо найти производные $y^{(n)}(x_0)$, n=0,1,2,3, но это можно сделать с помощью самого уравнения и заданных начальных условий.

Действительно, из начальных условий сразу же определяются значения функции и ее первой производной в заданной точке $x=x_0$. Вторая производная решения в этой же точке находится из самого уравнения:

$$y''(x_0) = F(x_0, y_0, y'_0)$$

Дифференцируя обе части исходного уравнения по переменной x, получим выражение для третьей производной решения:

$$y'''(x) = \left(\frac{\partial}{\partial x} F(x, y, y')\right) + \left(\frac{\partial}{\partial y} F(x, y, y')\right) y' + \left(\frac{\partial}{\partial y'} F(x, y, y')\right) y''$$

Подставляя в правую часть $x=x_0$ и учитывая уже известные значения соответствующих производных в этой точке, определяем значение третьей производной при $x=x_0$.

Дифференцируя выражение для третьей производной, получим представление четвертой производной решения через соответствующие частные производные известной функции правой части исходного дифференциального уравнения и предыдущие производные неизвестного ее решения. Подставляя в полученное выражение $x=x_0$ и учитывая найденные значения в этой точке предыдущих производных решения, находим значение четвертой производной решения в этой точке и т. д.

Найденные значения производных подставляем в представление решения в форме ряда Тейлора. Для тех значений x, для которых этот ряд сходится, он представляет искомое решение задачи Коши.

Для удобства использования мы разработали процедуру решения задачи Коши рассмотренного общего нелинейного дифференциального уравнения второго порядка, в которой реализовали описанный выше алгоритм построения ряда Тейлора решения. Ее текст представлен в примере 10.5.

Пример 10.5. Решение задачи Коши с помощью рядов

```
> intDiff:=proc(f::anything,x::name,y::name,y1::name,
              x0::numeric,y0::numeric,y01::numeric,
              n::numeric)
         local p,der,i,j,s;
         m0||1:=y01;
         m0||2:=subs(x=x0,y=y0,y1=y01,f);
         der:=subs(y1=m1,f);
```

```
          p:=y0+m0||1*(x-x0)+m0||2*(x-x0)^2/2;
          for i from 3 to n do
# Формирование дополнительных членов при дифференцировании производной
             s:=0;
             for j from 1 to i-2 do
                   s:=s+diff(der,m||j)*m||(j+1);
             end do;
# Формирование i-й производной решения
             der:=diff(der,x)+diff(der,y)*m||1+s;
# Вычисление значения i-й производной решения в точке x=x0
             m0||i:=eval(der,{x=x0,y=y0,seq(m||k=m0||k,k=1..i-1)});
# Добавление очередного члена ряда
             p:=p+(m0||i)*(x-x0)^i/i!;
          end do;
          end proc:
```

Передаваемыми параметрами в процедуру `intDiff()` являются выражение для правой части дифференциального уравнения (`f`) и используемые в нем неизвестные для представления независимой переменной (`x`), неизвестной функции решения (`y`) и ее производной (`y1`). Кроме этих величин, задающих дифференциальное уравнение, формальными параметрами `x0`, `y0` и `y01` определяется задача Коши: точка `x=x0`, в которой решение и его производная принимают соответственно значения `y0` и `y01`. Последний параметр `n` задает порядок усечения ряда, т. е. последний вычисляемый член ряда будет иметь вид $a_n(x-x_0)^n$.

Операторы, расположенные до начала цикла `for` по переменной `i`, формируют первые три члена ряда, коэффициенты которых вычисляются непосредственно из начальных условий, а также простой подстановкой начальных условий в правую часть дифференциального уравнения. Отметим, что в процедуре для всех появляющихся при последовательном дифференцировании правой части неизвестных производных решения используются неизвестные величины `mN`, где `N` — порядок производной, а для их вычисляемых значений в точке $x=x_0$ — переменные `m0N`.

В цикле `for` вычисляются оставшиеся неизвестные производные решения и их значения в точке начального условия, а также по полученным величинам формируется очередной член ряда и добавляется к переменной `p`, которая используется для формирования требуемого решения в виде степенного ряда.

Теперь продемонстрируем использование разработанной процедуры для построения решения нескольких задач Коши.

Задача 10.2

Найти решение уравнения

$$y'' = -y\,x^2\,,$$

удовлетворяющее начальным условиям

$$y(0)=1, y'(0)=0$$

Решение 1. Вызовем разработанную нами процедуру `intDiff()`:

```
> intDiff(-y*x^2,x,y,y1,0,1,0,20);
```

$$1-\frac{1}{12}x^4+\frac{1}{672}x^8-\frac{1}{88704}x^{12}+\frac{1}{21288960}x^{16}-\frac{1}{8089804800}x^{20}$$

Обратим внимание на то, что наша процедура работает, даже если правая часть не зависит от первой производной. Итак, решение построено, но правильное ли оно. Давайте попробуем решить нашу задачу Коши командой `dsolve()`:

```
> dsolve({diff(y(x),x$2)=-x^2*y(x),y(0)=1,D(y)(0)=0},y(x));
```

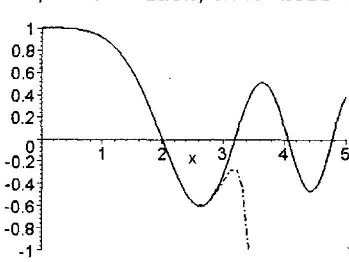
$$y(x)=\frac{1}{2}\Gamma\left(\frac{3}{4}\right)\sqrt{x}\ \mathrm{BesselJ}\left(\frac{1}{4},\frac{1}{2}x^2\right)-\frac{1}{2}\Gamma\left(\frac{3}{4}\right)\sqrt{x}\ \mathrm{BesselY}\left(\frac{1}{4},\frac{1}{2}x^2\right)$$

```
> s1:=rhs(%);
```

$$s1 := \frac{1}{2}\Gamma\left(\frac{3}{4}\right)\sqrt{x}\ \mathrm{BesselJ}\left(\frac{1}{4},\frac{1}{2}x^2\right)-\frac{1}{2}\Gamma\left(\frac{3}{4}\right)\sqrt{x}\ \mathrm{BesselY}\left(\frac{1}{4},\frac{1}{2}x^2\right)$$

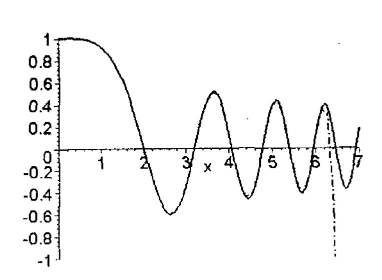
Точное решение выражается через специальные функции: гамма-функцию (`Γ(x)`) и функции Бесселя (`BesselJ()` и `BesselY()`). Для сравнения двух решений построим их графики:

```
> plot([s1,s],x=0..5,-1..1,color=black,thickness=2,linestyle=[1,4]);
```



На графике точное решение представлено сплошной линией, а построенное нами приближенное решение — в виде ряда пунктирной линией. Как видим, в интервале [`0, 2.8`] оба решения полностью совпадают, но далее, с увеличением независимой переменной, приближенное решение совершенно не соответствует точному. Это связано с количеством членов ряда в приближенном решении. Если увеличить их число с 20 до 100, то полученный ряд будет прекрасно аппроксимировать точное решение уже на интервале [`0, 6.1`], но далее опять будет резко убывать, отходя от точного решения:

```
> s:=intDiff(-y*x^2,x,y,y1,0,1,0,100):
> plot([s1,s],x=0..7,-1..1,color=black,thickness=2,linestyle=[1,4]);
```



Решение 2. Построить решение нелинейного дифференциального уравнения в форме ряда в Maple можно и другим способом, который при ручных вычислениях рекомендуется только для линейных уравнений. В этом способе решение y(x) представляется в форме ряда по степеням (x-x_0) с неопределенными коэффициентами

$$y(x) = a_0 + a_1\,(x - x_0) + a_2\,(x - x_0)^2 + a_3\,(x - x_0)^3 + a_4\,(x - x_0)^4 + \ldots,$$

которое подставляется в левую и правую части уравнения, причем все функции правой части также раскладываются в степенные ряды. Далее выполняются необходимые арифметические действия над рядами правой части, которая представляется в виде одного степенного ряда. После этого выписывается система линейных уравнений для вычисления неизвестных коэффициентов ряда решения, приравнивая коэффициенты при одинаковых степенях ($x-x_0$) левой и правой частей уравнения.

Как и в первом способе решения, мы разработали процедуру `intDiff1()` решения задачи Коши для дифференциального уравнения второго порядка с такими же формальными параметрами, как и в процедуре `intDiff()`. Полный текст этой процедуры представлен в примере 10.6.

Пример 10.6. Решение задачи Коши разложением решения в ряд

```
> intDiff1:=proc(f::anything,x::name,y::name,y1::name,
            x0::numeric,y0::numeric,y01::numeric,
            n::numeric)
         local p,pL,i,s,a,result;
         p:=y0+y01*(x-x0);
# Формируем решение с неопределенными коэффициентами a[i]
         result:=p+sum(a[i]*(x-x0)^i,i=2..n);
# Рабочий ряд (количество членов на 2 больше результирующего)
         p:=p+sum(a[i]*(x-x0)^i,i=2..n+2);
# Разложение правой части уравнения в ряд
         pL:=series(subs(y=p,y1=diff(p,x),f),x=x0,n+1);
# Формирование системы линейных уравнений
         for i from 0 to n do
```

```
                eq||i:=coeff(diff(p,x$2)-convert(pL,polynom),x,i)=0;
         end do;
# Решение полученной системы
         s:=solve({seq(eq||i,i=0..n)},{seq(a[i],i=2..n+2)});
         assign(s);
         eval(result);
         end proc:
```

В процедуре `intDiff1()` для того, чтобы правильно сформировать систему уравнений в соответствии с заданным усечением ряда `n`, введено рабочее решение `p`, в котором число членов на 2 больше требуемого, так как его приходится дважды дифференцировать. Однако результирующий ряд решения `result` содержит требуемое количество членов.

Теперь решим задачу 10.2 новым способом:

```
> s:=intDiff1(-y*x^2,x,y,y1,0,1,0,20);
```

$$s := 1 - \frac{1}{12}x^4 + \frac{1}{672}x^8 - \frac{1}{88704}x^{12} + \frac{1}{21288960}x^{16} - \frac{1}{8089804800}x^{20}$$

```
> s:=intDiff(-y*x^2,x,y,y1,0,1,0,20);
```

$$s := 1 - \frac{1}{12}x^4 + \frac{1}{672}x^8 - \frac{1}{88704}x^{12} + \frac{1}{21288960}x^{16} - \frac{1}{8089804800}x^{20}$$

Здесь же мы привели решение по старому способу, вызвав процедуру `intDiff()`. Результаты, как и ожидалось, совершенно идентичны, так как представление решения в виде ряда единственно и не важно, каким способом оно получено.

Задача 10.3

Найти решение уравнения

$$x\,y'' + y' + x\,y = 0\ ,$$

удовлетворяющее начальным условиям

$$y(0) = 1, y'(0) = 0$$

Решение. Чтобы решить это уравнение с помощью разработанных нами процедур, следует выразить вторую производную как функцию независимой переменной x, неизвестной функции y и ее первой производной:

$$y'' = -\frac{y'}{x} + y$$

Однако в этом случае функция правой части полученного дифференциального уравнения в точке x=0 не определена, поэтому применить процедуру, реализующую первый способ построения решения в виде ряда не представляется возможным, так как реализованный в нем алгоритм предусматривает вычисление правой части дифференциального уравнения в точке x=0 зада-

ния начальных условий, но в алгоритме, реализованном в процедуре `intDiff1()`, значение правой части в этой точке не вычисляется:

```
> s:=intDiff(-y1/x-y,x,y,y1,0,1,0,12);
Error, (in intDiff) division by zero

> s:=intDiff1(-y1/x-y,x,y,y1,0,1,0,12);
```

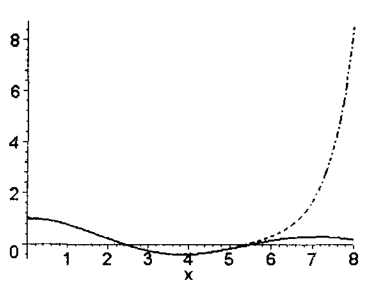
$$s := 1 - \frac{1}{4}x^2 + \frac{1}{64}x^4 - \frac{1}{2304}x^6 + \frac{1}{147456}x^8 - \frac{1}{14745600}x^{10} + \frac{1}{2123366400}x^{12}$$

Посмотрим, как согласуется построенное нами решение с точным решением задачи Коши:

```
> s1:=dsolve({x*diff(y(x),x$2)+diff(y(x),x)+x*y(x)=0,y(0)=1,D(y)(0)=0},
           y(x));
```

$$s1 := y(x) = \text{BesselJ}(0, x)$$

```
> plot([rhs(s1),s],x=0..8,color=black,thickness=2,linestyle=[1,4]);
```



Как и в предыдущей задаче, количество удержанных членов ряда не достаточно для представления решения на всем интервале его существования.

10.4. Ряды Фурье

В математике кроме степенных рядов широко используется разложение периодических функций в ряды Фурье. В Maple не существует специального пакета для построения подобного разложения функций. Однако, учитывая, что все необходимые коэффициенты ряда Фурье функции представляются через интегралы от произведения раскладываемой функции на тригонометрические функции синуса и косинуса различной периодичности, разработать процедуру разложения функции в ряд Фурье сравнительно легко. Напомним необходимые формулы.

Рядом Фурье периодической с периодом $2l$ функции $f(x)$ называется тригонометрический ряд вида

$$\frac{1}{2}a_0 + \left(\sum_{n=1}^{\infty} (a_n \cos(n\,x) + b_n \sin(n\,x))\right),$$

в котором коэффициенты вычисляются по следующим формулам:

$$a_0 = \frac{\int_{-l}^{l} f(x)\,dx}{l}$$

$$a_n = \frac{\int_{-l}^{l} f(x)\cos\left(\frac{n\,\pi\,x}{l}\right)dx}{l}$$

$$b_n = \frac{\int_{-l}^{l} f(x)\sin\left(\frac{n\,\pi\,x}{l}\right)dx}{l}$$

Для удобства работы разработаем процедуру разложения в ряд Фурье алгебраического выражения. Ее параметрами будут само выражение, имя независимой переменной, по которой выражение раскладывается в ряд Фурье, значение полупериода l и количество удерживаемых членов n. Процедура достаточна проста — командой `sum()` вычисляются конечные суммы, входящие в выражение ряда Фурье функции, ее текст представлен в примере 10.7.

Пример 10.7. Разложение периодической функции в ряд Фурье

```
> fourieSeries:=proc(f::algebraic,x::name,l::anything,n::nonnegint)
           (sum(int(f*cos(k*Pi*x/l),x=-l..l)*cos(k*Pi*x/l),k=0..n)+
           sum(int(f*sin(k*Pi*x/l),x=-l..l)*sin(k*Pi*x/l),k=1..n))/l;
           end proc:
```

В процедуре `fourieSeries()` раскладываемая в ряд функция задается в виде алгебраического выражения `f`, ее независимая переменная задается вторым параметром `x`, который должен быть не вычисленным именем. Параметр `l`, определяющий половину периода функции, можно задавать как в виде конкретного числа, так и в форме константы, например, `Pi`, или неопределенной величины, что позволяет получать разложение функции в ряд Фурье при произвольном неизвестном периоде $2l$. Последний параметр `n` определяет количество удерживаемых членов в ряде Фурье и не может быть отрицательным.

Если функция задается в виде процедуры, то в этом случае наша процедура разложения в ряд Фурье немного изменится — первый параметр должен быть типа procedure, а в теле процедуры вместо выражения `f` следует использовать обращение к функции `f(x)` (пример 10.8).

Пример 10.8. Разложение периодической функции, заданной в виде процедуры

```
> fourieSeries1:=proc(f::procedure,x::name,l::anything,n::nonnegint)
      (sum(int(f(x)*cos(k*Pi*x/l),x=-l..l)*cos(k*Pi*x/l),k=0..n)+
      sum(int(f(x)*sin(k*Pi*x/l),x=-l..l)*sin(k*Pi*x/l),k=1..n))/l;
      end proc:
```

Покажем, как с помощью разработанных процедур можно раскладывать в ряд Фурье кусочно-непрерывные функции и исследовать сходимость полученных рядов.

Задача 10.4

Разложить в ряд Фурье периодическую функцию f(x) с периодом 2π, которая определена следующим образом:

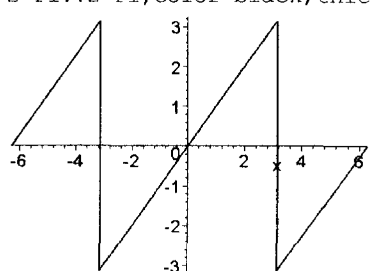
$$f(x)=x,\ -\pi < x \le \pi.$$

Решение. Прежде всего давайте построим график этой кусочно-непрерывной функции:

```
> f:=piecewise(x>-3*l and x<-l,x+2*l,x>-l and x<l,x,x>l and x<3*l,x-2*l);
```

$$f := \begin{cases} x+2\,l & -3\,l-x<0 \text{ and } x+l<0 \\ x & -l-x<0 \text{ and } x-l<0 \\ x-2\,l & l-x<0 \text{ and } x-3\,l<0 \end{cases}$$

```
> plot(eval(f,l=Pi),x=-2*Pi..2*Pi,color=black,thickness=2);
```



Для вычисления семи членов ряда Фурье заданной функции обратимся к процедуре `fourieSeries()`:

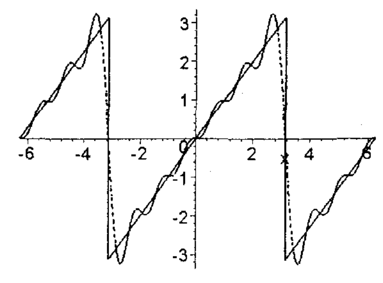
```
> normal(fourieSeries(eval(f,l=Pi),x,Pi,6));
```

$$2\sin(x)-\sin(2\,x)+\frac{2}{3}\sin(3\,x)-\frac{1}{2}\sin(4\,x)+\frac{2}{5}\sin(5\,x)-\frac{1}{3}\sin(6\,x)$$

Здесь нам пришлось вычислить выражение `f` при `l=Pi`, а также воспользоваться командой `normal()` для сокращения получаемого в результате вычисления процедурой выражения для ряда Фурье. Обратите внимание, так как функция нечетная, то ее ряд Фурье не содержит членов с косинусами.

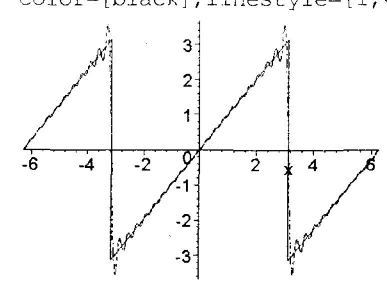
Теперь для оценки точности приближения заданной функции ее рядом Фурье с шестью членами построим графики полученного ряда и самой функции:

```
> plot([eval(f,l=Pi),fourieSeries(eval(f,l=Pi),x,Pi,6)],x=-2*Pi..2*Pi,
      numpoints=600, color=[black],linestyle=[1,4],thickness=2);
```

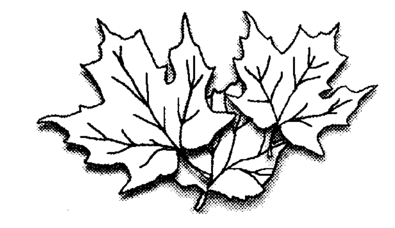


Видно, что удержанных в ряде Фурье членов не достаточно для приемлемого для практики приближения заданной пилообразной функции. Увеличение числа удерживаемых членов ряда Фурье позволит более точно аппроксимировать нашу функцию:

```
> plot([eval(f,l=Pi),fourieSeries(eval(f,l=Pi),x,Pi,20)],x=-2*Pi..2*Pi,
      numpoints=600, color=[black],linestyle=[1,4]);
```



Из построенного графика ряда Фурье с количеством членов 20 видно, что он уже достаточно хорошо приближает функцию во внутренних точках интервалов непрерывности функции, но в окрестности точек разрывов ряд начинает сильно осциллировать и его значения могут значительно отличаться от значения самой функции.



ГЛАВА 11

Численно-аналитические методы

Хотя Maple и является системой аналитических вычислений, но наличие в ней языка программирования, возможность работы с вещественными числами с плавающей точкой и развитая система графики делает ее прекрасным средством для быстрой реализации и визуализации разнообразных численных методов, начиная от простого уточнения методом Ньютона решения нелинейного уравнения и заканчивая сложными численно-аналитическими методами решения уравнений математической физики. Возможности Maple позволяют не только получить численное рушение поставленной задачи, но и достаточно быстро исследовать возможности самого метода, изменяя некоторые параметры задачи.

11.1. Исследование метода Ньютона

Итерационный метод Ньютона в силу его быстрой сходимости очень часто используется для уточнения решения нелинейного уравнения вида

$$f(x)=0.$$

Для его корректного применения необходимо определить интервал изменения переменной, на котором уравнение имеет точно один корень, а начальное приближение x_0 выбирать из условия выполнения следующего неравенства:

$$f(x_0)\cdot f'(x_0) > 0.$$

Сами приближения вычисляются по следующей формуле:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Как обычно, разработаем процедуру, строящую числовую последовательность, предел которой равняется корню функции, на основе расчетных формул метода Ньютона (пример 11.1).

Пример 11.1. Решение нелинейного уравнения методом Ньютона

```
> Newton:=proc(f::algebraic,x::name,x0::float,EPS::float)
            local i,z,s;
            z||0:=x0;
            z||1:=z||0-eval(f/diff(f,x),x=z||0);
            s:=[x0,eval(f,x=x0)],[z||1,eval(f,x=z||1)];
            for i from 2 while
                  abs(evalf(z||(i-1))-evalf(z||(i-2)))>EPS do
                        z||i:=z||(i-1)-eval(f/diff(f,x),x=z||(i-1));
                        s:=s,[z||i,eval(f,x=z||i)];
            end do;
            end proc:
```

Параметрами процедуры `Newton()` являются левая часть уравнения `f`, задаваемая в виде алгебраического выражения Maple, имя неизвестной переменной функции, корень которой ищется, а также начальное приближение `x0` и точность `EPS` в виде вещественных чисел с плавающей точкой. В самой процедуре в переменных `zi`, где `i` является числом, представляющим номер итерации, вычисляются соответствующие приближения к корню уравнения. Возвращаемым значением этой процедуры является последовательность двухэлементных списков, представляющих приближение к корню и значение функции в точке приближенного решения. Такое возвращаемое значение в форме последовательности списков предусмотрено нами специально с целью последующей графической демонстрации построения приближающей последовательности.

Задача 11.1

Найти методом Ньютона приближенное решение нелинейного уравнения

$$x^3-x=0.$$

Замечание

Конечно, для рассматриваемого уравнения корни можно определить и непосредственным разложением левой части на множители, но мы выбрали это уравнение исключительно в связи с тем, что для него графическая интерпретация метода Ньютона достаточно наглядна.

Решение. Прежде всего необходимо отделить корни уравнения. Для нашего уравнения это сделать достаточно просто. Его корнями являются точки $x=0$, $x=1$ и $x=-1$, поэтому соответствующие интервалы, в которых расположено

по одному корню уравнения, определяются легко. Будем искать приближение к корню $x=1$ на интервале [0,52; 2] изменения переменной x.

Дальше следует выбрать начальное приближение из условия выполнения приведенного в начале раздела неравенства. Для нашего интервала такой точкой может быть точка $x=2$:

$$f(2)\cdot f'(2) = 6/11 > 0.$$

Найдем с помощью разработанной процедуры `Newton()` приближение к корню с точностью до 0,01:

```
> f:=x^3-x:
> s:=Newton(f,x,2.0,0.01);
```

s := [2.0, 6.000], [1.454545454, 1.622839967], [1.151046789, .373985128],
[1.025325929, .052592310], [1.000908452, .001819381], [1.000001235, .2470 10^{-5}]

Как видим, для получения решения с заданной точностью оказалось достаточным выполнить всего 6 итераций.

Метод Ньютона имеет простую геометрическую интерпретацию: очередное приближение находится как точка пересечения оси абсцисс и касательной к графику функции в точке предыдущего приближения. Для визуализации построения итерационного процесса разработаем процедуру, отображающую график функции, а также касательную к нему в каждой точке приближающей последовательности. Полный текст подобной процедуры представлен в примере 11.2.

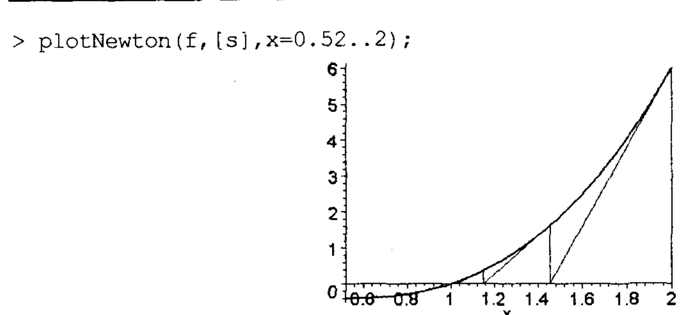
Пример 11.2. Процедура визуализации метода Ньютона

```
> plotNewton:=proc(f::algebraic,s::list(list),x::equation)
                local i;
                p||0:=plot(f,x,thickness=2,color=black);
                for i to nops(s)-1 do
                        p||i:=
                        plot([[op(1,op(i,s)),0],op(i,s),
                        [op(1,op(i+1,s)),0]],x,color=black);
                end do;
                `plots/display`({seq(p||i,i=0..nops(s)-1)});
                end proc:
```

В процедуру `plotNewton()` передается выражение левой части нелинейного уравнения, список двухэлементных списков, состоящих из точки последовательности приближения и значения функции в этой точке, а также имя переменной и диапазон ее изменения в виде уравнения `x=a..b`. Результатом выполнения этой процедуры будет график нелинейной функции, а также множество касательных в точках приближения.

Для нашей задачи обращение к процедуре `plotNewton()` построит следующий график:

```
> plotNewton(f,[s],x=0.52..2);
```



Практически после третьего шага мы попадаем в малую окрестность решения, в которой построенные касательные уже просто сливаются с графиком самой функции.

Обратимся теперь к нахождению второго корня $x=0$, расположенного на интервале [-0,5; 0,52]. Проверим выполнение условия положительности произведения функции на ее производную на правом конце интервала:

$$f(0{,}52)\cdot f'(0{,}52) \approx 2{,}0049 > 0.$$

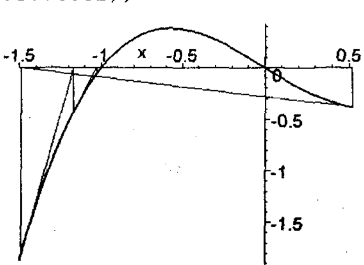
Следовательно, эту точку можно взять за начальное приближение:

```
> s:=Newton(f,x,0.52,0.01);
```

s := [.52, -.379392], [-1.489491525, -1.815072035], [-1.168566727, -.427167458], [-1.030621462, -.064084659], [-1.001312780, -.002630732], [-1.000002577, -.5154 10^{-5}]

Удивительно, но мы почему-то получили приближение к решению $x=-1$. Что произошло? Дело в том, что для сходимости последовательности метода Ньютона с начальным значением, удовлетворяющим неравенству $f(x_0)\cdot f'(x_0) > 0$, необходимо еще выполнение условия, чтобы на интервале поиска решения первая производная функции $f(x)$ не меняла своего знака. В нашем случае на интервале [-0,5; 0,52] первая производная меняет свой знак, поэтому пересечение касательной к графику функции в точке начального приближения происходит не на заданном интервале, а вне интервала, а отсюда последующие приближения оказываются приближением к другому корню:

```
> plotNewton(f,[s],x=-1.5..0.52);
```



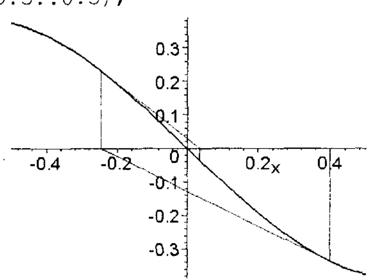
Чтобы получить приближающую последовательность к необходимому нам корню, следует взять другое начальное приближение, например, x=0,4:

```
> s:=Newton(f,x,0.4,0.01);
```

$s := [.4, -.336], [-.2461538462, .2312389623], [.0364566877, -.03640823348],$

$[-.00009729639, .00009729638908], [.184\ 10^{-11}, -.184\ 10^{-11}]$

```
> plotNewton(f,[s],x=-0.5..0.5);
```



Обратите внимание, что в этом случае последовательность приближенных значений корня стремится к своему пределу не с одной стороны, как в случае с первым корнем x=1, а с двух сторон. Это связано с тем, что вторая *производная в окрестности корня* x=0 *меняет знак.*

Как видите, применение метода Ньютона требует выполнение целого ряда условий, иначе мы либо получим другой корень, либо приближающая последовательность окажется не имеющей вообще предела, как, например, при нахождении корня того же уравнения $x^3-x=0$ при начальном условии $x_0=\pm\frac{1}{5}\sqrt{5}$:

```
> s:=Newton(f,x,evalf(sqrt(5)/5,20),0.01);
```

$s := [.44721359549995793928, -.3577708764], \%1, \%2, \%1, \%2, \%1, \%2, \%1, \%2, \%1$

$\%1 := [-.4472135955, .3577708764]$

$\%2 := [.4472135955, -.3577708764]$

```
> plotNewton(f,[s],x=-0.5..0.5);
```

Как видно из построенного рисунка, в данном случае касательная к графику функции в точке первого приближения пересекается с осью абсцисс в точке нулевого приближения, образуя тем самым замкнутый цикл.

Замечание

Для того чтобы наша процедура `Newton()` **построения приближающей последовательности к решению не зациклилась, нам пришлось в ее цикл** `for` **добавить оператор**

```
if i > 10 then return s; end if;
```

Если использовать метод Ньютона без предварительного анализа существования решения уравнения и определения интервала, на котором оно существует и единственно, то последовательность, построенная по методу Ньютона, может сходиться, но, естественно, не к решению уравнения. Пример подобной функции можно найти в [5]:

```
> p:=evalf(ln(2))/evalf(Pi)*sin(2*evalf(Pi*ln(x)/ln(2)))+1;
```

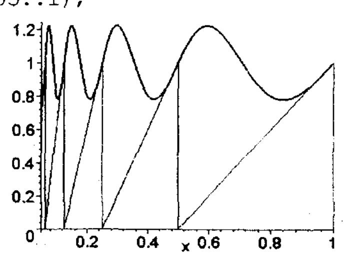
$$p := .2206356001\ \sin(9.064720284\ \ln(x)) + 1$$

```
> Digits:=20:s:=Newton(p,x,1.0,0.0001);
```

$s := [1.0, 1.], [.49999999989974560708, .99999999954612690944],$
$[.25000000001321387920, .99999999999999999577],$
$[.12499999998154334190, .99999999954612691371],$
$[.062500000006606939598, .99999999999999999133],$
$[.031249999997037570512, .99999999954612691831],$
$[.015625000002477602346, .99999999999999998666],$
$[.0078124999996723263874, .99999999954612692291],$
$[.0039062500008258674485, .99999999999999998222],$
$[.0019531250000213150366, .99999999954612692722],$
$[.00097656250025808357771, .99999999999999997777],$
$[.00048828125003113711910, .99999999954612693167],$
$[.00024414062507742507334, .99999999999999997355],$
$[.00012207031251423636975, .99999999954612693605],$
$[.000061035156272582313056, .99999999999999996910]$

Как видим, при начальном приближении x_0=1 последовательность метода Ньютона сходится к нулю, однако функция вообще не имеет корней, что видно из ее графика, на котором также демонстрируется получение последовательности с помощью касательных:

```
> plotNewton(p,[s],x=0.05..1);
```



Приведенное исследование решения простого нелинейного уравнения методом Ньютона показывает, что нельзя без предварительного анализа задачи применять к ней численные методы решения, так как можно получить либо не то решение, которое ожидалось, либо вообще его не получить, либо получить совершенно неправильный ответ.

11.2. Интерполирование функций полиномами

Очень часто на практике в результате измерения какой-либо величины получают ее значения на некотором дискретном множестве точек ее области изменения. В дальнейшем может оказаться необходимым получить ее значения в точках, не равных узловым точкам, в которых получены значения функции. Для решения подобной задачи по известным значениям функции строится другая функция, которая аппроксимирует исходную в соответствии с некоторым выбранным критерием. Если в качестве критерия выбирается равенство значений исходной и аппроксимирующей функций в узловых точках, то построенная в соответствии с этим критерием функция называется интерполирующей. Если функция, более того, выбирается из полиномов, то такая аппроксимирующая функция называется интерполяционным полиномом. Так как для однозначного определения полинома степени n требуется знание его значений в $n+1$ точках, то и для однозначного построения интерполяционного полинома функции необходимо знать ее значения в $n+1$ точках.

Для построения интерполяционного полинома $\mathrm{L}(x)$ степени n функции $\mathrm{f}(x)$, заданной в $n+1$ не равных между собой узловых точках x_0, x_1, ..., x_n, очень часто применяется формула Лагранжа:

$$\mathrm{L}(x)=\sum_{i=0}^{n}\mathrm{f}(x_i)\prod_{\substack{j=0\\ j\neq i}}^{n}\frac{(x-x_j)}{(x_i-x_j)}.$$

Хотя Maple и содержит в своем составе команду `interp()`, строящую интерполяционный полином для функции, мы разработаем собственную процедуру его построения, тем более что это достаточно простое дело — всего лишь запрограммировать приведенную выше формулу. Текст процедуры интерполяции функции представлен в примере 11.3.

Пример 11.3. Интерполяция функции, заданной в узловых точках

```
> interpolation:=proc(x::vector,y::vector,v::name)
      local i,j,s,mult,n;
# Определение размерности вектора x
      n:=`linalg/vectdim`(x);
```

```
# Определение размерности вектора y и сравнение с размерностью вектора x
      if n <> `linalg/vectdim`(y) then
      error "Векторы %1 и %2 должны быть одинаковой размерности",x,y
      end if;
# Вычисление суммы
      s:=0;
      for i from 1 to n do
# Вычисление произведений в сумме
            mult:=1;
            for j from 1 to n do
                  if j=i then
                        continue;
                  else
                        mult:=mult*(v-x[j])/(x[i]-x[j]);
                  end if;
            end do;
            s:=s+mult*y[i];
      end do;
      sort(simplify(s));
      end proc:
```

Входными параметрами в процедуре `interpolation()` являются два вектора, содержащие координаты узловых точек и значений интерполируемой функции в них, а также имя переменной, которое используется в качестве независимой переменной интерполяционного полинома. Размерности векторов, которые определяются в теле процедуры, должны совпадать, иначе процедура не будет работать и напечатает сообщение об ошибке. Если их размерности совпадают, то далее в двойном цикле строится интерполяционный полином степени на единицу меньше размерностей векторов узловых точек и значений функции, который и является возвращаемым значением процедуры.

Построим интерполяционный полином функции, заданной таблицей своих значений

x_i	0	0.5	1.2	1.4	2.2	3
$f(x_i)$	2.5	5.6	6.8	2.9	0.0	10.0

Так как функция задана в 6 точках, то интерполяционный полином должен иметь степень, равную пяти. Построим его с помощью разработанной нами процедуры `interpolation()`:

```
> x:=vector(6,[0,0.5,1.2,1.4,2.2,3]):
> y:=vector(6,[2.5,5.6,6.8,2.9,0.0,10]):
> s:=interpolation(x,y,z);
```

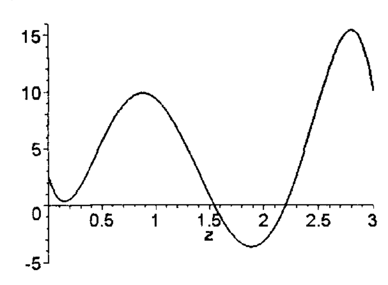
$s := -9.185424796\, z^5 + 65.66931876\, z^4 - 156.8421884\, z^3 + 138.9006106\, z^2 -$

$31.67433396\,z + 2.499999999$

```
> sort(interp(x,y,z));
```

$s := -9.185424809\,z^5 + 65.66931884\,z^4 - 156.8421886\,z^3 + 138.9006107\,z^2 -$
$31.67433402\,z + 2.499999999$

```
> plot(s,z=0..3,-5..16,color=black,thickness=2);
```



Как видим, коэффициенты построенного по нашей процедуре интерполяционного полинома незначительно отличаются от полинома, построенного встроенной в Maple командой `interp()`. Такое незначительное отличие связано с тем, что при вычислениях по встроенной команде используется большее число значащих цифр в мантиссе, чем при вычислениях по нашей процедуре. Этот недочет можно исправить, добавив в начало тела процедуры `interpolation()` оператор `Digits:=30`, а при выходе из нее восстановить количество значащих цифр в мантиссе по умолчанию, выполнив оператор `Digits:=10`.

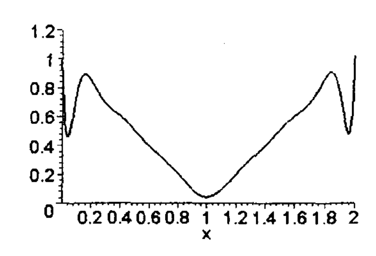
Интерполяционный полином хорошо аппроксимирует гладкую функцию, имеющую непрерывные производные до порядка $n+1$ включительно. Однако если аппроксимируемая функция имеет разрывную производную какого-либо порядка, то интерполяционный полином на концах отрезка интерполирования начинает сильно осциллировать и совершенно не пригоден для аппроксимации функции между узлами, расположенными ближе к концам интервала аппроксимации.

Построим интерполяционный полином функции, заданной своими значениями в 16 точках на интервале [0, 2] в соответствии со следующей таблицей:

i	1	2	3	4	5	6	7	8
x_i	0	0.13	0.27	0.4	0.53	0.67	0.8	0.93
$f(x_i)$	1	0.87	0.73	0.6	0.47	0.33	0.2	0.07
i	9	10	11	12	13	14	15	16
x_i	1.07	1.2	1.33	1.47	1.6	1.73	1.87	2
$f(x_i)$	0.07	0.2	0.33	0.47	0.6	0.73	0.87	1

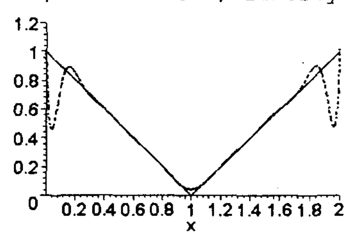
Прежде всего создадим два вектора x и y размерности 16, в которых будем хранить, соответственно, координаты узлов и значения функции в них, а затем построим интерполяционный полином (в приводимом ниже тексте программы мы опустили присваивание элементам векторов соответствующих значений):

```
> t:=vector(16):y:=vector(16):
> s:=interpolation(t,y,x):
> plot(s, x=0..2,0..1.2, numpoints=100,
        scaling=CONSTRAINED, thickness=2, color=black);
```



Видно, что график интерполяционного полинома очень похож на график функции y=abs(x-1), но только на концах интервала интерполирования он резко отличается от него. Действительно, данные для построения интерполяционного полинома соответствуют указанной функции. Чтобы сравнить графики этих двух функций, мы приводим их на одном рисунке:

```
> plot([abs(x-1),s], x=0..2, 0..1.2, numpoints=100,
      scaling=CONSTRAINED, thickness=2, linestyle=[1,4], color=black);
```



В данном случае мы аппроксимировали интерполяционным полиномом функцию, у которой первая производная имеет разрыв в точке $x=1$. Результат такого разрыва сказался на поведении интерполяционного полинома в областях изменения независимой переменной, примыкающих к концевым точкам интервала интерполирования. Если интерполируемая функция не имеет достаточного числа гладких производных, то нельзя утверждать, что ее интерполяционный полином будет с достаточной степенью точности ее аппроксимировать, в чем мы сейчас и убедились.

Теперь обратимся к интерполированию функций двух переменных. Maple для подобного случая не предоставляет пользователю никаких команд. Од-

нако запрограммировать алгоритм построения интерполяционного полинома для функции двух переменных в Maple не представляет труда.

Прежде всего, приведем необходимые расчетные формулы. Будем рассматривать функцию двух переменных $f(u,v)$, заданную таблицей своих значений $f(u_i, v_j) = f(x^{ij}) \equiv y_{ij}$ в узловых точках $x^{ij} = (u_i, v_j)$:

	v_1	v_2	...	v_m
u_1	y_{11}	y_{12}	...	y_{1m}
u_2	y_{21}	y_{22}	...	y_{2m}
...	...	...	...	...
u_n	y_{n1}	y_{n2}	...	y_{nm}

Возьмем произвольную строчку i таблицы и построим интерполяционный полином Лагранжа степени $m-1$ относительно переменной v, используя числа y_{ij}, $j=1..m$ этой строки в качестве значений в узлах, определяемых верхней строкой таблицы:

$$P_{m-1,i}(v) = \sum_{\mu=0}^{m-1} a_{i,\mu} v^{\mu}$$

Мы записали его в стандартном виде, к которому может быть приведен любой полином. Первый индекс $m-1$ в обозначении полинома P указывает на его степень. Этот полином в силу своего построения обладает свойством

$$P_{m-1,i}(v_j) = y_{i,j}, \quad j = 1..m$$

Получив такие полиномы для всех i=1..n, построим еще серию полиномов Лагранжа степени $n-1$ относительно переменной u, используя в качестве узлов значения из крайней левой колонки таблицы значений исходной функции, а значениями в этих узловых точках будут, соответственно, коэффициенты $a_{i,\mu}$, i=1..n:

$$K_{n-1,\mu}(u) = \sum_{\nu=0}^{n-1} b_{\mu,\nu} u^{\nu}, \quad K_{n-1,\mu}(u_i) = a_{i,\mu}, \quad i = 1..n$$

Построив полиномы $K_{n-1,\mu}(u)$ для μ=0..m-1, тем самым получим требуемый интерполяционный полином

$$L_{n-1,m-1}(u,v) = \sum_{\mu=0}^{m-1} K_{n-1,\mu}(u) v^{\mu}$$

или, подставляя вместо $K_{n-1,\mu}(u)$ его представление в виде суммы,

$$L_{n-1,m-1}(u,v) = \sum_{\mu=0}^{m-1} \left(\sum_{\nu=0}^{n-1} b_{\mu,\nu} u^{\nu} \right) v^{\mu},$$

который в заданных узловых точках (u_i, v_j) принимает значения y_{ij} из таблицы.

Теперь в соответствии с приведенным алгоритмом разработаем процедуру `interp3D()` построения интерполяционного полинома функции двух переменных, заданной своими значениями в узлах прямоугольной сетки. Ее полный текст приведен в примере 11.4.

Пример 11.4. Интерполяция функции двух переменных

```
> interp3D:=proc(x::vector,y::vector,A::matrix,u::name,v::name)
            local i,j,s,m,n,UV,F,r;
# Определение размерностей сетки значений
            m:=`linalg/rowdim`(A);
            n:=`linalg/coldim`(A);
# Здесь следует вставить проверку соответствия размерностей
# векторов узлов x и y и матрицы значений A
            UV:=matrix(m,n):
# Построение интерполяционных полиномов по строкам матрицы A
# и сохранение их коэффициентов в возрастающем порядке
# в столбцах матриц UV
            for j from 1 to m do
                  F:=interp(y,row(A,j),v);
                  for i from 1 to n do
                        UV[j,i]:=coeff(F,v,i-1);
                  end do;
            end do:
# Построение интерполяционных полиномов по коэффициентам
# построенных интерполяционных полиномов
# и сохранение их коэффициентов в возрастающем порядке
# в матрице r
            r:=matrix(m,n):
            for j from 1 to n do
                  F:=interp(x,col(UV,j),u);
                  for i from 1 to m do
                        r[i,j]:=coeff(F,u,i-1);
                  end do;
            end do:
# Построение интерполяционного полинома табличной функции
            s:=0;
            for i from 1 to m do
                  for j from 1 to n do
                        s:=s+r[i,j]*u^(i-1)*v^(j-1);
                  end do;
            end do;
            simplify(s);
            end proc:
```

В процедуру `interp3D()` передаются векторы `x` и `y` узлов сетки по двум координатным осям, прямоугольная матрица `A` значений функции в узлах заданной сетки, а также имена переменных `u` и `v`, относительно которых строится интерполяционный полином. Для построения интерполяционных полиномов одной переменной в процедуре используется команда `interp()` из стандартной библиотеки Maple, хотя можно было бы строить их и разработанной нами процедурой `interpolation()`.

Построим интерполяционный полином функции двух переменных, заданной таблицей своих значений на следующей сетке (матрица `F` хранит значения функции в узлах сетки):

```
> x:=vector(3,[1,2,3]);
```

$$x := [1, 2, 3]$$

```
> y:=vector(3,[0,2,3]);
```

$$y := [0, 2, 3]$$

```
> F:=matrix(3,3,[ [ 1,2, 0],
                  [ 0,3, 9],
                  [-2,0,-3]
                ]);
```

$$F := \begin{bmatrix} 1 & 2 & 0 \\ 0 & 3 & 9 \\ -2 & 0 & -3 \end{bmatrix}$$

```
> s:=interp3D(x,y,F,u,v);
```

$$s := 1 + \frac{44}{3}v - \frac{25}{3}v^2 + \frac{1}{2}u - \frac{203}{12}u\,v + \frac{121}{12}u\,v^2 - \frac{1}{2}u^2 + \frac{53}{12}u^2\,v - \frac{31}{12}u^2\,v^2$$

Теперь проверим, правильно ли построен интерполяционный полином `s`, вычислив его значения в узлах сетки:

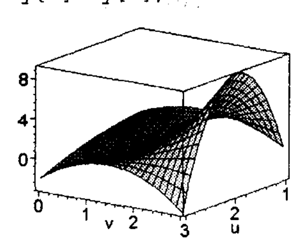
```
> ff:=matrix(3,3):
  for i to 3 do
       for j to 3 do
             ff[i,j]:=eval(s,{u=x[i],v=y[j]})
       end do:
  end do:
  evalm(ff);
```

$$\begin{bmatrix} 1 & 2 & 0 \\ 0 & 3 & 9 \\ -2 & 0 & -3 \end{bmatrix}$$

Результат, как и следовало ожидать, полностью совпадает с исходной матрицей `F` значений функции в узлах сетки.

Теперь остается только построить график интерполяционного полинома, чтобы оценить его поведение:

```
> with(plots):
> plot3d(s,u=x[1]..x[3],v=y[1]..y[3],axes=BOXED,orientation=[35,70]);
```



11.3. Краевые задачи для обыкновенных дифференциальных уравнений

В этом разделе мы покажем применение Maple для построения приближенного *аналитического* решения краевой задачи для линейного обыкновенного дифференциального уравнения второго порядка. Подобные методы появились задолго до возникновения компьютерной техники, но с ее появлением были несколько оттеснены в сторону численными методами, для которых компьютер является идеальным средством реализации, так как работает именно с числовым представлением функций, а не их аналитическим выражением. Системы аналитических вычислений позволяют достаточно просто реализовывать не только численные решения разнообразных задач, но и строить их приближенные аналитические решения.

Из большого количества приближенных аналитических методов мы выбрали метод Галеркина, который в настоящее время переживает, скажем так, свое второе рождение в связи с широким внедрением в практику расчета разнообразных конструкций метода конечных элементов, основные разрешающие уравнения которого могут быть выведены с помощью метода Галеркина.

Сформулируем общую краевую задачу для обыкновенного дифференциального уравнения второго порядка: найти решение дифференциального уравнения

$$Lu \equiv u'' + \mathrm{p}(x)\, u' + \mathrm{q}(x)\, u = \mathrm{f}(x), \qquad a \le x \le b,$$

удовлетворяющее двум краевым условиям

$$l_0 u \equiv \alpha_0 u\,(a) + \beta_0 u'(a) = \gamma_0,$$
$$l_1 u \equiv \alpha_1 u(b) + \beta_1 u'(b) = \gamma_1,$$

где p, q и f являются непрерывными функциями переменной x на интервале ее изменения $[a, b]$, а α_i, β_i и γ_i — некоторые заданные числа, причем некоторые из них могут равняться нулю, но не все одновременно.

Для нахождения приближенного решения краевой задачи на отрезке $[a, b]$ задают некоторую линейно независимую систему дважды непрерывно дифференцируемых функций $\varphi_0, \varphi_1, ..., \varphi_n, ...$ таких, что функция φ_0 удовлетворяет заданным краевым условиям

$$l_0\varphi_0 = \gamma_0,$$
$$l_1\varphi_0 = \gamma_1,$$

а остальные функции системы при $i=1, 2, ...$ удовлетворяют однородным краевым условиям

$$l_0\varphi_0 = 0,$$
$$l_1\varphi_0 = 0.$$

Эта система функций называется базисной.

Приближенное решение краевой задачи ищется в виде линейной комбинации $n+1$ базисных функций

$$y_n(x) = \varphi_0(x) + a_1\varphi_1(x) + \ldots + a_n\varphi_n(x)$$

с неизвестными коэффициентами a_i, $i=1..n$, которые находятся из выполнения некоторого критерия. Следует отметить, что построенное таким способом приближенное решение в силу выбора базисных функций точно удовлетворяет краевым условиям задачи.

В методе Галеркина в качестве критерия для определения неопределенных коэффициентов приближенного решения выбирается требование ортогональности базисных функций $\varphi_1(x), \varphi_2(x), ..., \varphi_n(x)$ к функции невязки $\Psi(x; a_1, ..., a_n)$, представляющей разность левой и правой частей обыкновенного дифференциального уравнения при подстановке в него приближенного решения

$$\Psi(x; a_1, ..., a_n) = Ly_n - f(x) = L\varphi_0(x) - f(x) + \sum_{k=1}^{n} a_k L\,\varphi_n(x).$$

Под ортогональностью понимается равенство нулю скалярного произведения невязки и базисных функций:

$$\int_a^b \Psi(x; a_1, \ldots, a_n)\,\varphi_i(x)\,dx = 0.$$

Это требование приводит к линейной системе алгебраических уравнений для определения неизвестных коэффициентов a_i приближенного решения y_n:

$$a_1(L\varphi_1, \varphi_1) + \ldots + a_n(L\varphi_n, \varphi_1) = (f - L\varphi_0, \varphi_1),$$
$$a_1(L\varphi_1, \varphi_2) + \ldots + a_n(L\varphi_n, \varphi_2) = (f - L\varphi_0, \varphi_2),$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$a_1(L\varphi_1, \varphi_n) + \ldots + a_n(L\varphi_n, \varphi_n) = (f - L\varphi_0, \varphi_n).$$

Решение этой системы всегда существует и единственно в силу линейной независимости базисных функций.

Разработаем процедуру построения приближенного решения краевой задачи для обыкновенного дифференциального уравнения второго порядка, в которой в качестве параметров будем передавать дифференциальный оператор уравнения L, правую часть, процедуру вычисления базисных функций и количество n удерживаемых в приближенном решении базисных функций. Ее текст представлен в примере 11.5.

Пример 11.5. Процедура метода Галеркина

```
> Galerkin:=proc(L::procedure,f::algebraic,phi::procedure,x::name,
                a::numeric,b::numeric,n::numeric)
            local y,i,eq,A;
            y:=phi(0,x)+sum(A[i]*phi(i,x),i=1..n);
            for i to n do
      eq||i:=sum(A[k]*int(L(phi(k,x),x)*phi(i,x),x=a..b),k=1..n)=
                  int((f-L(phi(0,x),x))*phi(i,x),x=a..b);
            end do;
            eq:=solve({seq(eq||i,i=1..n)},{seq(A[i],i=1..n)});
            assign(eq);
            simplify(eval(y));
            end proc:
```

В процедуру `Galerkin()` дифференциальный оператор L уравнения передается в виде процедуры с двумя параметрами, первый из которых представляет собой алгебраическое выражение, а второй задает имя независимой переменной, по которой происходит дифференцирование. Правая часть дифференциального уравнения передается в виде алгебраического выражения `f`. Параметр-процедура `phi()` вычисляет n-ю базовую функцию φ_n, и ее параметры должны представлять номер базовой функции и имя независимой переменной, используемой в формировании выражения базовой функции. Параметр `x` процедуры `Galerkin()` определяет имя независимой переменной. Оставшиеся три ее параметра задают координаты граничных точек и удерживаемое число членов n в приближенном решении.

Задача 11.2

Решить методом Галеркина краевую задачу

$$Lu \equiv u'' + u' = -x, \quad 0 \le x \le 1, \quad u(0) = 0, \quad u(1) = 0.$$

Решение. Прежде всего выбираем базисные функции

$$\varphi_0(x) = 0, \quad \varphi_i(x) = x^i(1-x), \quad i = 1, 2, \ldots$$

Функция $\varphi_0(x)$ удовлетворяет граничным условиям краевой задачи, а функции $\varphi_i(x)$, $i=1..n$ — однородным граничным условиям.

Теперь необходимо создать процедуры для вычисления дифференциального оператора уравнения и базовых функций, а также задать выражение для правой части уравнения:

```
> L1:=proc(y,x)
      diff(y,x$2)+y;
      end proc:
> f:=-x:
> phi:=proc(n,x)
      if n=0 then 0 else x^n*(1-x) end if;
      end proc:
```

Итак, все необходимые процедуры созданы, и можно переходить к построению приближенного решения:

```
> res1:=Galerkin(L1,f,phi,x,0,1,1);
```

$$res1 := -\frac{5}{18}x(-1+x)$$

```
> res2:=Galerkin(L1,f,phi,x,0,1,2);
```

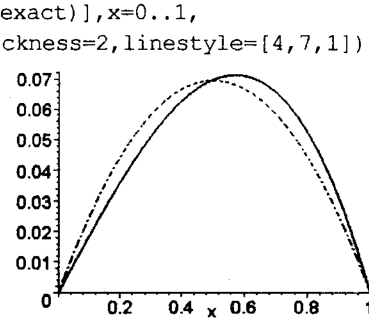
$$res2 := \frac{71}{369}x - \frac{8}{369}x^2 - \frac{7}{41}x^3$$

Здесь мы построили два приближенных решения, в которых удержано, соответственно, один и два члена решения. Посмотрим, как они согласуются с точным решением краевой задачи, а мы специально выбрали такую задачу, для которой возможно построение точного решения:

```
> exact:=dsolve({diff(y(x),x$2)+y(x)=-x,y(0)=0,y(1)=0},y(x));
```

$$exact := y(x) = -x + \frac{\sin(x)}{\sin(1)}$$

```
> plot([res1,res2,rhs(exact)],x=0..1,
      color=black,thickness=2,linestyle=[4,7,1]);
```



Видно, что уже приближенное решение, в котором удержаны две базовые функции (n=2), на графике сливается с точным решением, и только первое

приближение, график которого представлен пунктирной линией, существенно отличается от точного.

Конечно, второе приближение все-таки отличается от точного. Чтобы увидеть это, мы представили в табл. 11.1 числовые значения точного решения и трех первых приближений к нему в десяти точках промежутка [0, 1].

Таблица 11.1. *Сравнение приближенных и точного решений*

x	Первое приближение	Второе приближение	Третье приближение	Точное решение
.0	0	0	0	0
.1	.250000000e-1	.188536585e-1	.186252304e-1	.186415438e-1
.2	.444444444e-1	.362493225e-1	.361054354e-1	.360976604e-1
.3	.583333333e-1	.511626016e-1	.512195693e-1	.511947674e-1
.4	.666666666e-1	.625691056e-1	.628027734e-1	.627828522e-1
.5	.694444444e-1	.694444444e-1	.697463768e-1	.697469638e-1
.6	.666666666e-1	.707642277e-1	.709978954e-1	.710183520e-1
.7	.583333333e-1	.655040651e-1	.655610327e-1	.655851467e-1
.8	.444444444e-1	.526395665e-1	.524956794e-1	.525024677e-1
.9	.250000000e-1	.311463415e-1	.309179134e-1	.309018658e-1
1.0	0	0	0	0

Из табл. 11.1 видно, что у первого приближения погрешность порядка 0,01, у второго — порядка 0,001, а третье уже имеет точность порядка 0,0001.

Если необходимо решить другую краевую задачу, то для этого необходимо изменить функцию $\varphi_0(x)$ таким образом, чтобы она удовлетворяла новым граничным условиям. Например, если граничные условия исходной задачи заменить на следующие

$$u\,(0) = 1, \quad u\,(1) = 0,$$

то можно выбрать функцию $\varphi_0(x) = (1-x)$. После этого следует изменить процедуру `phi()` и построить приближенные решения:

```
> phi:=proc(n,x)
      x^n*(1-x);
      end proc:
> res1:=Galerkin(L1,f,phi,x,0,1,1);
```

$$res1 := 1 - \frac{4}{9}x - \frac{5}{9}x^2$$

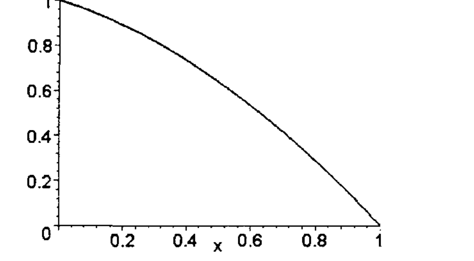
```
> res2:=Galerkin(L1,f,phi,x,0,1,2);
```

$$res2 := 1 - \frac{4}{9}x - \frac{5}{9}x^2$$

```
> exact:=dsolve({diff(y(x),x$2)+y(x)=-x,y(0)=1,y(1)=0},y(x));
```

$$exact := y(x) = -x + \cos(x) - \frac{(-1 + \cos(1))\sin(x)}{\sin(1)}$$

```
> plot([res1,res2,rhs(exact)],x=0..1,
      color=black,thickness=2,linestyle=[4,7,1]);
```



Для этой задачи график уже первого приближения полностью совпадает с графиком точного решения.

Задача 11.3

Решить методом Галеркина краевую задачу

$$Lu \equiv u'' + (1 + x^2)\, u' = -1, \quad -1 \le x \le 1, \quad u\,(-1) = 0, \quad u\,(1) = 0.$$

Решение. Задаемся базисными функциями

$$\varphi_0\,(x) = 0, \quad \varphi_i\,(x) = x^{2i-2}\,(1-x^2), \quad i = 1, 2, \ldots$$

Создаем необходимые процедуры вычисления оператора и правой части дифференциального уравнения, а также базисных функций:

```
> L1:=proc(y,x)
      diff(y,x$2)+(1+x^2)*y;
      end proc:
> f:=-1:
> phi:=proc(n,x)
      if n=0 then 0 else x^(2*n-2)*(1-x^2); end if;
      end proc:
> res1:=Galerkin(L1,f,phi,x,-1,1,1);
```

$$res1 := \frac{35}{38} - \frac{35}{38}x^2$$

```
> res2:=Galerkin(L1,f,phi,x,-1,1,2);
```

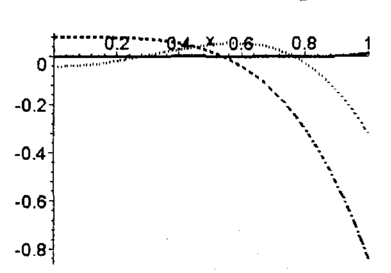
$$res2 := \frac{3969}{4252} - \frac{1050}{1063}x^2 + \frac{231}{4252}x^4$$

```
> res3:=Galerkin(L1,f,phi,x,-1,1,3);
```

$$res3 := \frac{5148231}{5523436} - \frac{62073}{64226}x^2 + \frac{26169}{5523436}x^4 + \frac{81939}{2761718}x^6$$

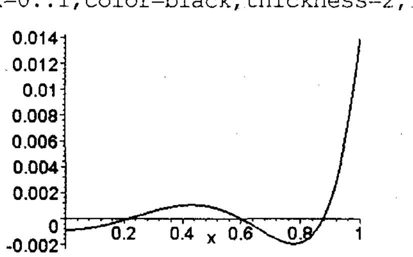
К сожалению, точного решения для поставленной краевой задачи Maple найти не смог. Поэтому, чтобы оценить погрешность полученных приближенных решений, мы построим графики их невязок:

```
> plot([L1(res1,x)-f,L1(res2,x)-f,L1(res3,x)-f],x=0..1,
          color=black,thickness=3,linestyle=[4,7,1]);
```



Невязка первого приближения отображается штрихпунктирной кривой, второго приближения — точечной кривой, а третьего приближения сплошной кривой, практически совпадающей с осью абсцисс. Чтобы оценить точность третьего приближения, отобразим отдельно график ее невязки:

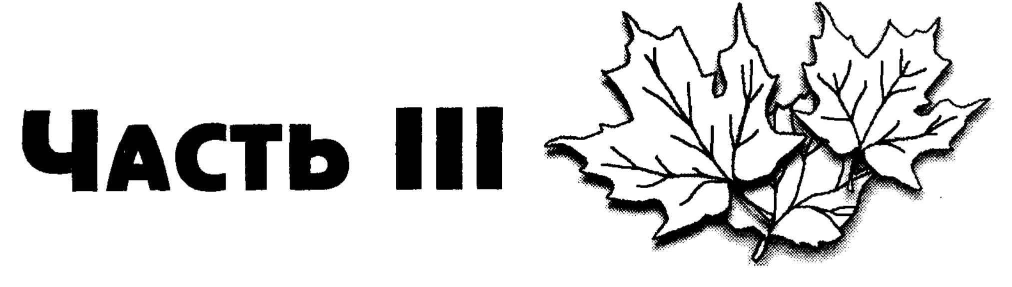
```
> plot([L1(res3,x)-f],x=0..1,color=black,thickness=2,linestyle=1);
```



Видно, что максимальное отклонение невязки третьего приближения от нуля составляет 0,014.

Решение этой задачи показывает эффективность построения приближенных решений краевых задач обыкновенных дифференциальных уравнений методом Галеркина. Однако следует сказать об основной трудности применения этого метода, заключающейся в выборе линейно независимой системы базисных функций, удовлетворяющей однородным краевым условиям.

Часть III



Механика

Глава 12. Задачи теоретической механики

Глава 13. Метод начальных параметров в расчете балок

Глава 14. Задачи теории упругости

Третья часть книги посвящена решению задач теоретической механики, строительной механики и механики деформируемого твердого тела.

Из задач теоретической механики рассмотрены задачи кинематики и динамики точки, входящие в учебные курсы по данному предмету.

Задачи расчета строительных конструкций в настоящее время в основном решаются на основе численных методов — с использованием разнообразных разностных схем или конечно-элементного подхода. Однако многие практически важные задачи, например расчет балок переменного сечения, можно решить на основе численно-аналитического подхода, который был несколько отодвинут в сторону в связи с широким внедрением в расчетную практику численных методов. Системы аналитических вычислений могут в корне изменить отношение к "забытым" аналитическим методам решения задач строительной механики и механики деформируемого твердого тела.

Из обширной области механики, изучающей поведение деформируемых твердых тел, решены две классические задачи теории упругости. В первой задаче средствами Maple исследуется напряженное состояние в точке упругого тела, а во второй — решение в рядах Фурье изгиба тонкой шарнирно опертой изотропной пластинки.



ГЛАВА 12

Задачи теоретической механики

В кинематике рассматривается движение тел без учета их массы и действующих на них сил. Во многих задачах можно пренебречь размерами тела и рассматривать его как материальную точку. Все характеристики движения точки (скорость, ускорение, пройденный путь) могут быть получены из уравнения ее движения, задаваемого в параметрической форме. В качестве параметра в задачах механики выступает время. Одной из первых, решаемых в разделе кинематики точки задач и является задача определения траектории, скорости и ускорения точки, заданной своими уравнениями движения.

Задача 12.1

По заданным уравнениям движения точки

$$x = 4 - 2\,t\,,$$

$$y = 2 + 2\cos\left(\frac{1}{4}\pi\,t\right)$$

установить вид ее траектории и для момента времени t=1 с найти положение точки на траектории, ее скорость, полное, касательное и нормальное ускорения, а также радиус кривизны траектории в соответствующей точке.

Решение. Уравнения движения являются параметрическими уравнениями траектории точки. Для получения уравнения траектории в обычной координатной форме следует исключить время t из уравнений движения:

```
> eq1:=x=4-2*t;eq2:=y=2+2*cos(Pi/4*t);
```

$$eq1 := x = 4 - 2\,t$$

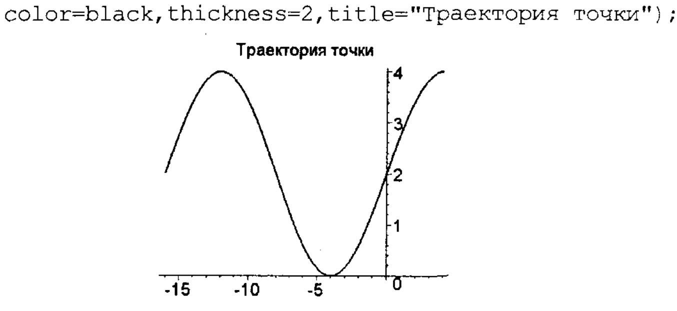
$$eq2 := y = 2 + 2\cos\left(\frac{1}{4}\pi\,t\right)$$

```
> simplify(subs(isolate(eq1,t),eq2));
```

$$y = 2 + 2\cos\left(\frac{1}{8}\pi\,(x-4)\right)$$

Командой `isolate(eq1,t)` мы выразили из уравнения, определяющего закон изменения координаты x точки, параметр t и подставили его в уравнение изменения координаты y. График траектории представляет часть синусоиды:

```
> plot([rhs(eq1),rhs(eq2),t=0..10],
          color=black,thickness=2,title="Траектория точки");
```



Для определения скорости точки находим проекции скорости на оси координат дифференцированием по времени уравнений движения:

```
> v[x]:=diff(rhs(eq1),t);eval(%,t=1);
```

$$v_x := -2$$

$$-2$$

```
> v[y]:=diff(rhs(eq2),t);eval(%,t=1);evalf(%);
```

$$v_y := -\frac{1}{2}\sin\left(\frac{1}{4}\pi\,t\right)\pi$$

$$-\frac{1}{4}\sqrt{2}\,\pi$$

$$-1.110720734$$

Модуль скорости точки в момент времени t=1 с вычисляется по формуле

$$v = \sqrt{{v_x}^2 + {v_y}^2}$$

и равен

```
> v:=sqrt(v[x]^2+v[y]^2);evalf(eval(%,t=1));
```

$$v := \frac{1}{2}\sqrt{16 + \sin\left(\frac{1}{4}\pi\,t\right)^2\pi^2}$$

$$2.287728251$$

Аналогично находятся проекции ускорения точки на оси координат как первые производные от проекций скоростей и модуль ускорения в заданной точке траектории:

```
> w[x]:=diff(v[x],t);evalf(eval(%,t=1));
```

$$w_x := 0$$

$$0.$$

```
> w[y]:=diff(v[y],t);evalf(eval(%,t=1));
```

$$w_y := -\frac{1}{8}\cos\left(\frac{1}{4}\pi\, t\right)\pi^2$$

$$-.8723580250$$

```
> W:=sqrt(w[x]^2+w[y]^2);evalf(eval(%,t=1));
```

$$W := \frac{1}{8}\pi^2\sqrt{\cos\left(\frac{1}{4}\pi\, t\right)^2}$$

$$.8723580250$$

Касательное ускорение определяется дифференцированием модуля скорости:

$$w_\tau = \left|\frac{dv}{dt}\right|,$$

$$\frac{dv}{dt} = \frac{(v_x w_x + v_y w_y)}{v}$$

В момент времени, заданный в условиях задачи, он будет равен:

```
> w[tau]:=(v[x]*w[x]+v[y]*w[y])/V;evalf(eval(w[tau],t=1));
```

$$w_\tau := \frac{1}{8}\frac{\sin\left(\frac{1}{4}\pi\, t\right)\pi^3\cos\left(\frac{1}{4}\pi\, t\right)}{\sqrt{16+\sin\left(\frac{1}{4}\pi\, t\right)^2\pi^2}}$$

$$.4235407532$$

Знак w_τ положителен, а это означает, что движение точки ускоренное и, следовательно, направления векторов скорости и касательного ускорения в данный момент времени совпадают.

Нормальное ускорение точки вычисляется по формуле

$$w_n = \sqrt{w^2 - w_\tau^{\ 2}}$$

и в момент времени t=1 с будет равен:

```
> w[n]:=sqrt(W^2-w[tau]^2);evalf(eval(%,t=1));
```

$$w_n := \frac{1}{8}\sqrt{\cos\left(\frac{1}{4}\pi\, t\right)^2\pi^4 - \frac{\sin\left(\frac{1}{4}\pi\, t\right)^2\pi^6\cos\left(\frac{1}{4}\pi\, t\right)^2}{16+\sin\left(\frac{1}{4}\pi\, t\right)^2\pi^2}}$$

$$.7626413012$$

Для вычисления радиуса кривизны траектории можно воспользоваться формулой

$$\rho = \frac{v^2}{w_n},$$

которая в заданный момент времени дает следующую величину нормального ускорения:

```
> rho:=V^2/w[n]:evalf(eval(%,t=1));
                              6.862597846
```

Итак, все характеристики движения точки вычислены, хотя, справедливости ради, следует заметить, что в этой части решения задачи Maple использовался нами всего лишь как калькулятор. Однако когда мы перейдем к построению графика траектории и векторов скорости и ускорения в заданный момент, то вот здесь Maple незаменим. Можно разработать процедуру, которая строит траекторию точки по заданным уравнениям движения, а также в каждой точке траектории векторы скорости и всех ускорений (общего, касательного и нормального), причем эта процедура будет создавать анимационную картинку, на которой пользователь увидит движущуюся точку с соответствующими векторами. Текст этой процедуры представлен в примере 12.1.

Пример 12.1. Движение точки по заданной траектории

```
> move:=proc(x::anything, y::anything, t::name, tn::numeric,
      xx::numeric, a::numeric, b::numeric, c::numeric, d::numeric,
      widthV::numeric, widthW::numeric)
      local vx,vy,wx,wy,w,w1,wtau,wn,s,w1n,w1t,
      t1,t2,t3,t4,t5,t6,g,v,f,v1,dt,ff,i,roo;
# Вычисление скорости
      vx:=diff(x,t);
      vy:=diff(y,t);
      v:=sqrt(vx^2+vy^2);
# Вычисление ускорения
      wx:=diff(vx,t);
      wy:=diff(vy,t);
      w:=sqrt(wx^2+wy^2);
      wtau:=(vx*wx+vy*wy)/v;
      wn:=sqrt(w^2-wtau^2);
# Вычисление радиуса кривизны
      roo:=v^2/wn;
      dt:=tn/40;
# Вычерчивание траектории
      ff:=plot([x,y,t=0..2*tn],color=black,thickness=3);
      for i from 0 to 40 do
```

```
# Построение вектора скорости
      v1:=`plottools/arrow`(
             [eval(x,t=i*dt),eval(y,t=i*dt)],
             vector([eval(vx,t=i*dt),eval(vy,t=i*dt)]),
             widthV,3*widthV,0.1,color=green);
# Построение вектора ускорения
      w1:=`plottools/arrow`(
             [eval(x,t=i*dt),eval(y,t=i*dt)],
             vector([eval(wx,t=i*dt),eval(wy,t=i*dt)]),
             widthW,3*widthW,.1,color=red);
# Построение вектора касательного ускорения
      w1t:=plot([[eval(x,t=i*dt),eval(y,t=i*dt)],
             [eval(x+wtau*vx/v,t=i*dt),eval(y+wtau*vy/v,t=i*dt)]],
             color=pink,thickness=4);
# Построение вектора нормального ускорения
      w1n:=plot([[eval(x,t=i*dt),eval(y,t=i*dt)],
             [eval(x-wn*vy/v,t=i*dt),eval(y+wn*vx/v,t=i*dt)]],
             color=pink,thickness=4);
# Отображение координат вектора скорости
      s:=cat("v(x,y) = [",convert(evalf(eval(vx,t=i*dt),3),string),
             ",", convert(evalf(eval(vy,t=i*dt),3),string), "]");
             t1:=plots[textplot]([xx,a,s],align={ABOVE,RIGHT});
# Отображение координат вектора ускорения
      s:=cat("w(x,y) = [", convert(evalf(eval(wx,t=i*dt),3),string),
             ",", convert(evalf(eval(wy,t=i*dt),3),string), "]");
             t2:=plots[textplot]([xx,b,s],align={ABOVE,RIGHT});
# Отображение значений касательного и нормального ускорений
      s:=cat("w(tau,n) = [",
             convert(evalf(eval(wtau,t=i*dt),3),string), ",",
             convert(evalf(eval(wn,t=i*dt),3),string), "]");
             t3:=plots[textplot]([xx,c,s],align={ABOVE,RIGHT});
# Отображение значения радиуса кривизны
      s:=cat("ro = ", convert(evalf(eval(roo,t=i*dt),3),string));
             t4:=plots[textplot]([xx,d,s],align={ABOVE,RIGHT});
# Отображение момента времени
      s:=cat("t = ",convert(evalf(i*dt,3),string));
             t5:=plots[textplot]([0.2,0.4,s],align={ABOVE,RIGHT});
# Построение кадра анимации
      g||i:=`plots/display`({ff,v1,w1,w1t,w1n,t1,t2,t3,t4,t5});
      end do;
# Отображение анимации
      `plots/display`([seq(g||i,i=0..40)],
             insequence=true,scaling=constrained);
      end proc:
```

В процедуру `move()` примера 12.1 передаются выражения (`x` и `y`), задающие изменение во времени координат *x* и *y* точки. Параметр `t` задает имя переменной, используемой в этих выражениях в качестве переменной времени. Числовой параметр `tn` определяет момент времени, до которого отображается движение точки. Траектория на графике строится до момента `2*tn`.

Процедура вычисляет декартовы координаты векторов скорости и ускорения, а также значения касательного и нормального ускорений и радиуса кривизны траектории точки в дискретные моменты времени. Параметр `xx` задает координату *x*, а параметры `a`, `b`, `c` и `d` координаты *y* строк, отображающих значения указанных величин.

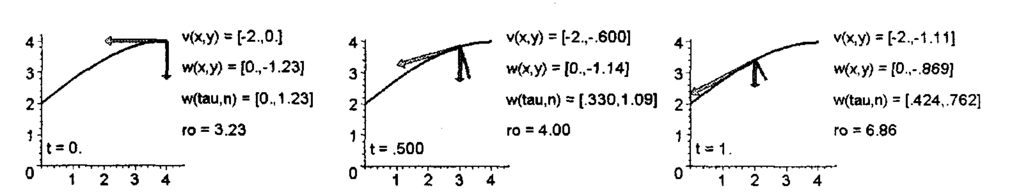
Величины `widthV` и `widthW` задают толщины стрелок, используемых для отображения векторов скорости и ускорения точки, в долях от их длин. Вектор скорости отображается зеленой стрелкой, а ускорения красной. Векторы касательного и нормального ускорений отображаются пурпуровыми линиями.

Замечание

Координаты точек привязки надписей, содержащих значения рассчитываемых векторов, как и толщины стрелок, используемых для отображения векторов скорости и ускорения, введены в связи с тем, что эту процедуру можно использовать для расчета произвольного движения точки, заданного уравнениями движения. Однако в общем случае достаточно трудно автоматизировать расположение надписей на чертеже, поэтому такая работа возлагается на пользователя процедуры.

Воспользуемся разработанной процедурой `move()` и создадим анимационную картинку движения точки по заданным в задаче уравнениям движения:

```
> move(4-2*t, 2+2*cos(Pi/4*t), t, 1, 4.5, 4, 3, 2, 1, 0.1, 0.1);
```



На рисунке представлены три кадра анимации, соответствующие начальному, промежуточному и конечному моментам времени движения точки. При запуске анимации на рабочем листе Maple точка будет перемещаться вдоль траектории из начального положения в конечное. Векторы скорости и ускорений будут изменяться вместе с движущейся точкой, отражая направление и величину ее скорости и ускорения в текущий момент времени.

Движение точки завершается в момент времени t=1 с, равный расчетному времени задачи. Значения всех вычисленных в процедуре характеристик

движения точки совпадают с их значениями, вычисленными нами ранее в начале раздела.

Процедура `move()` универсальна, и ее можно использовать для исследования движения точки с любыми заданными уравнениями движения. Например, рассчитаем характеристики движения точки, законы изменения координат которой имеют следующий вид:

```
> x:=cos(t/3);
```

$$x := \cos\left(\frac{1}{3}t\right)$$

```
> y:=3/2*sin(t/2)/2;
```

$$y := \frac{3}{4}\sin\left(\frac{1}{2}t\right)$$

```
> move(x,y,t,3.5,0.2,-0.25,-0.5,-0.75,-1,0.02,0.02);
```


Из приведенных трех кадров анимации видно, что сначала точка движется замедленно ($w_\tau < 0$), а затем ускоренно ($w_\tau > 0$). Момент времени, после которого точка начинает ускоряться, можно приближенно определить, просмотрев анимацию по кадрам, что дает значение t=2.62 с.

Внимание!

С помощью процедуры `move()` можно решать практически любую задачу на изучение движения точки по ее уравнениям движения. Правда, процедура может не работать для тех движений, в которых вектор скорости или ускорения принимает нулевое значение. В этом случае не будет запущена процедура построения стрелки `arrow()`. В связи с ограниченным объемом книги в нее не удалось включить исправленный вариант процедуры. Читателю предоставляется самому откорректировать процедуру `move()`.

Разработанную процедуру студент может использовать для проверки своих вычислений, а преподаватель — для разработки лабораторных работ.

Для задач динамики Maple легко позволяет визуализировать поведение системы. Рассмотрим его применение к построению и решению динамических уравнений движения материальной точки.

Задача 12.2

Два груза D и E массами $m_D = 2$ кг и $m_E = 10$ кг лежат на гладкой плоскости, наклоненной под углом α=30° к горизонту, опираясь на пружину, коэффициент жесткости которой $c = 600$ Н/м.

В некоторый момент времени груз E убирают; одновременно (t=0) нижний конец пружины B начинает совершать вдоль наклонной плоскости движение по закону $\xi = 0.2\sin(10t)$ м. Здесь ξ перемещение относительно начального положения точки прикрепления пружины B. Найти уравнение движения груза D.

Решение. Совместим начало координатной системы с положением покоя груза D, соответствующим статической деформации пружины, при условии, что точка B занимает свое среднее положение (ξ=0). Направим ось x вверх вдоль наклонной плоскости (рис. 12.1).

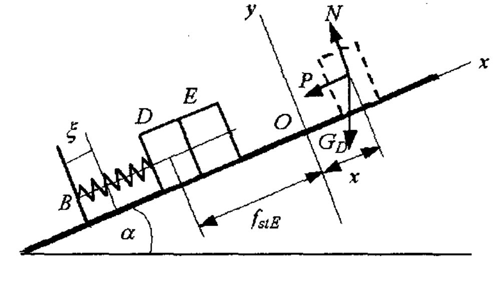


Рис. 12.1. Оси координат и силы, действующие на тело D

Движение груза D определяется по следующему дифференциальному уравнению:

$$m_D\ddot{x} = \sum_i X_i ,$$

где $\sum_i X_i$ — сумма проекций на ось x сил, действующих на груз D. В соответствии с рис. 12.1 такими силами являются: G_D — вес тела, N — нормальная реакция наклонной плоскости, равная $G_D\sin(\alpha)$, и P — сила упругости пружины, которая вычисляется по формуле:

$$P = -\,c\,(x - f_{\text{stD}} - \xi),$$

где f_{stD} — статическая деформация пружины под действием груза D, ξ — перемещение точки прикрепления нижнего конца пружины, происходящее по закону $\xi = d\sin(pt)$ ($d = 0.2$ м, $p = 10$ с$^{-1}$).

Определим на рабочем листе Maple дифференциальное уравнение движения тела D:

```
> restart:
> de:=mD*diff(x(t),t$2)=-G[D]*sin(alpha)-P;
```

$$de := mD\left(\frac{\partial^2}{\partial t^2}\,\mathrm{x}(t)\right) = -G_{\mathrm{D}}\sin(\alpha) - P$$

```
> P:=c*(x(t)-f[stD]-xi);
```

$$P := c\,(\mathrm{x}(t) - f_{stD} - d\sin(p\,t))$$

```
> xi:=d*sin(p*t);
```

$$\xi := d\sin(p\,t)$$

```
> de;
```

$$mD\left(\frac{\partial^2}{\partial t^2}\,\mathrm{x}(t)\right) = -G_{\mathrm{D}}\sin(\alpha) - c\,(\mathrm{x}(t) - f_{stD} - d\sin(p\,t))$$

Замечание

Все обозначения совпадают с введенными нами при выводе дифференциального уравнения, за исключением масс тел *D* и *E*, которые, соответственно, обозначены через `mD` и `mE`.

Статическая деформация пружины f_{stD} находится из уравнения покоя груза *D* на наклонной плоскости:

```
> st1:=-G[D]*sin(alpha)+P0=0;
```

$$st1 := -G_{\mathrm{D}}\sin(\alpha) + P0 = 0$$

```
> P0:=c*f[stD];
```

$$P0 := c\,f_{stD}$$

```
> f[stD]:=solve(st1,f[stD]);
```

$$f_{stD} := \frac{G_{\mathrm{D}}\sin(\alpha)}{c}$$

Теперь дифференциальное уравнение движения тела *D* будет выглядеть следующим образом:

```
> de:=expand(simplify(de));
```

$$de := mD\left(\frac{\partial^2}{\partial t^2}\,\mathrm{x}(t)\right) = -c\,\mathrm{x}(t) + c\,d\sin(p\,t)$$

В начальный момент времени положение тела *D* относительно выбранной системы отсчета равняется $x_0 = -f_{\mathrm{stE}}$, причем $f_{\mathrm{stE}} = (G_E\sin(\alpha))\,/\,c$ — статическая деформация пружины под действием груза *E*, а скорость $\dot{x}_0 = 0$.

Решим дифференциальное уравнение с учетом начальных условий:

```
> sol:=dsolve({de,x(0)=-f[stE],D(x)(0)=0},x(t));
```

$$sol := \mathrm{x}(t) = -\frac{c\,d\,p\,mD\sin\left(\frac{\sqrt{c\,mD}\;t}{mD}\right)}{\sqrt{c\,mD}\;(c - p^2\,mD)} - f_{stE}\cos\left(\frac{\sqrt{c\,mD}\;t}{mD}\right) + \frac{c\,d\sin(p\,t)}{c - p^2\,mD}$$

Теперь остается присвоить значения всем величинам и построить график движения тела *D*:

```
> alpha:=Pi/6; d:=0.2; p:=10; g:=9.81; c:=600;
  mD:=2; G[D]:=mD*g;mE:=10; G[E]:=mE*g;
  f[stE]:=G[E]*sin(alpha)/c;
```

$\alpha := \frac{1}{6}\pi$

$d := .2$

$p := 10$

$g := 9.81$

$c := 600$

$mD := 2$

$G_D := 19.62$

$mE := 10$

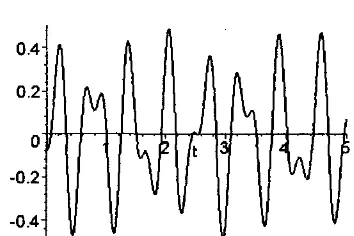
$G_E := 98.10$

$f_{stE} := .08175000000$

```
> evalf(sol);
```

$$x(t) = -.173\sin(17.3\,t) - .0818\cos(17.3\,t) + .300\sin(10.\,t)$$

```
> plot(rhs(sol),t=0..5,color=black,thickness=2);
```



Итак, решение получено. Теперь можно воспользоваться графическими возможностями Maple и создать анимационную картинку движения тела *D* по наклонной плоскости.

Для этого сначала создадим на рабочем листе две процедуры. Первая (пример 12.2) будет отображать тело *D* на наклонной плоскости в зависимости от значения координаты *x* его центра тяжести, а с помощью второй процедуры (пример 12.3) можно будет нарисовать пружину по координатам ее концов.

Пример 12.2. Отображение тела на наклонной плоскости

```
> body:=proc(x,alpha)
     `plottools/rotate`(
```

```
               `plottools/rectangle`([x-0.25,1],[x+0.25,0],color=gray),
               alpha,[0,0]);
end proc:
```

Пример 12.3. Отображение пружины

```
> strin:=proc(x,x1,alpha)
               local str,delt,points;
               delt:=abs(x-x1)/13;
               points:=plot([[x,0],[x,1],[x,0.5],
               seq([x+delt/2+delt*i,0.5+(-1)^i*0.2],i=0..12),
               [x+delt*13,0.5]],
               color=black,thickness=2,scaling=constrained);
               points:=`plottools/rotate`(points,alpha,[0,0]);
        end proc:
```

Обе процедуры возвращают графические PLOT-структуры. И тело, и пружина сначала рисуются на горизонтальной плоскости, а затем полученные графические образы командой `rotate()` пакета `plottools` поворачиваются вокруг начала координат на угол `alpha`, чтобы представить их расположенными на наклонной плоскости.

Процедура `pdif()`, представленная в примере 12.4, использует их для построения анимационной картинки движения тела и пружины по наклонной плоскости.

Пример 12.4. Движение тела по наклонной плоскости

```
> pdif:=proc(eq,eq1,alpha,T)
# eq    — уравнение движения центра тяжести тела
# eq1   — уравнение движения левого конца пружины
# alpha — угол наклонной плоскости
# T     — интервал в секундах отображения движения тела
        local l1,rec,i,dt,cadr,ps,pb,TT;
# количество кадров в анимации
        cadr:=T*10;
# отображение наклонной плоскости
               l1:=plot({0,tan(alpha)*x},x=0..6,
               thickness=3,color=black,axes=NONE);
# перпендикуляр, проходящий через точку начального
# положения левого конца пружины
        ps:=`plottools/rotate`(
               plot([[1,0],[1,1]],color=black),alpha,[0,0]);
# перпендикуляр, проходящий через точку начала координат
        pb:=`plottools/rotate`(
               plot([[3,0],[3,3]],color=black),alpha,[0,0]);
```

```
# интервал между кадрами
      dt:=T/cadr;
# создание кадров анимации
      for i from 0 to cadr do
# отображение момента времени, которому соответствует кадр
              TT:=`plots/textplot`([5,5,cat("t = ",
                     convert(evalf(dt*i,4),string))]);
# отображение тела на наклонной плоскости
              rec:=body(3+eval(eq,t=dt*i),alpha);
# непосредственное формирование кадра
              q||i:=`plots/display`({l1,rec,ps,pb,TT,
                     strin(eval(1+eq1,t=dt*i),
                     eval(3+eq-0.25,t=i*dt),
                     alpha)},
                     scaling=constrained);
      end do;
# отображение полученной анимации
      `plots/display`([seq(q||i,i=0..cadr)],
      insequence=true,scaling=constrained);
  end proc:
```

Воспользуемся разработанными процедурами и построим анимационную картинку движения тела *D* в соответствии с полученным решением его уравнения движения:

```
> pdif(rhs(sol),xi,alpha,5);
```

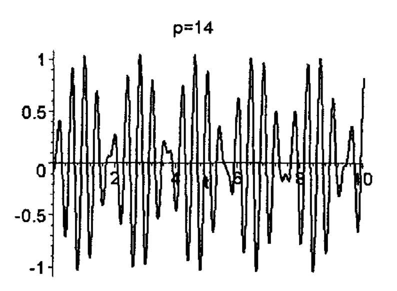
t = 0. t = 2. t = 5.

Здесь, как всегда при представлении анимации, мы показали ее три кадра. Так как колебательное движение тела, судя по построенному ранее графику изменения его положения на наклонной плоскости, происходит с большой частотой, то для получения реалистического движения тела следует увеличить в процедуре `pdif()` значение переменной `cadr` таким образом, чтобы на одну секунду движения приходилось не менее 50 кадров. В этом случае аппроксимация движения в построенной анимации будет более реалистичной и соответствующей действительному характеру колебания тела.

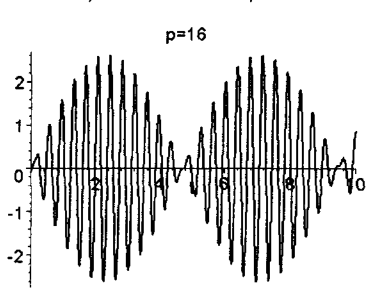
Полученное решение можно использовать для анализа поведения колебаний тела в зависимости от исходных параметров системы: жесткости пружины, массы тела *E* или частоты вынужденных колебаний левого конца пружины.

При приближении частоты вынужденных колебаний к собственной частоте колебаний системы, ее поведение будет приближаться к резонансному, которое характеризуется неограниченным возрастанием амплитуды колебаний тела *D*. Построим графики движения тела *D* при частоте вынуждающей силы *p*, равной 14, 16 и 17 $с^{-1}$. Для этого достаточно изменить на рабочем листе значение переменной `p`, так как решение `sol` получено для произвольных значений параметров системы:

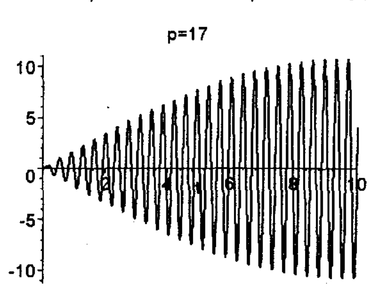
```
> p:=14:plot(rhs(sol),t=0..10,color=black,thickness=2,title="p=14");
```




```
> p:=16:plot(rhs(sol),t=0..10,color=black,thickness=2,title="p=16");
```




```
> p:=17:plot(rhs(sol),t=0..10,color=black,thickness=2,title="p=17");
```

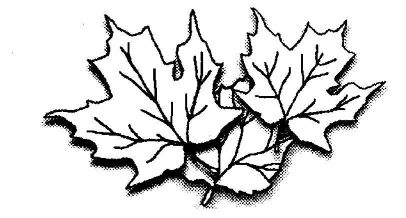


На рисунках можно наблюдать характер изменения колебательного движения тела при приближении частоты вынуждающей силы к резонансной частоте системы. Можно создать анимационную картинку, на которой удобнее наблюдать изменение и характера колебательного движения, и увеличения амплитуды колебаний:

```
> plots[animate](rhs(sol),t=0..10,p=14..17,
             frames=80,numpoints=300,color=black,thickness=2);
```

Внимание!

Чтобы команда создания анимации работала, нужно снова выполнить все команды Maple, с помощью которых мы строили решение задачи, но из блока команд присваивания необходимых числовых значений параметрам задачи следует удалить команду `p:=10`.



ГЛАВА 13

Метод начальных параметров в расчете балок

Метод начальных параметров расчета нагруженных балок основан на одном из старейших методов решения задачи Коши для дифференциального уравнения — разложении решения в ряд Тейлора в окрестности точки начального условия. Конечно, при ручном счете или при его реализации в традиционных системах программирования возникают значительные трудности при удержании достаточно большого числа членов ряда в решении или разложении переменных коэффициентов в ряды Тейлора, но реализация подобного подхода в системах аналитических вычислений достаточна проста и не требует больших усилий при программировании разложения в ряды любых функций. Реализация метода начальных параметров в Maple, с точки зрения автора, показывает всю мощь и силу систем аналитических вычислений в построении эффективных аналитических алгоритмов решения задач, математическая модель которых описывается системой дифференциальных уравнений.

Рассмотрим балку переменного сечения, лежащую на упругом основании переменной жесткости (рис. 13.1). Ее изгиб под воздействием внешних сил описывается обыкновенным дифференциальным уравнением четвертого порядка с переменными коэффициентами:

$$[E\,\mathrm{K}(x)\,\mathrm{w}''(x)]'' - T\,\mathrm{w}''(x) + \mathrm{k}(x)\,\mathrm{w}(x) = \mathrm{q}(x)\,.$$

В этом уравнении произведение $E{\cdot}\mathrm{K}(x)$ — жесткость балки, где E модуль упругости, а $\mathrm{K}(x)$ момент инерции поперечного сечения балки относительно оси, проходящей через его центр тяжести, $\mathrm{k}(x)$ — коэффициент постели упругого основания, $\mathrm{q}(x)$ — распределенная по всей длине балки нагрузка, T — величина осевого усилия, а $\mathrm{w}(x)$ — прогиб балки по оси z.

Замечание

Обычно момент инерции поперечного сечения балки относительно оси, проходящей через его центр тяжести, в строительной механике обозначается $\mathrm{I}(x)$, но

в силу того, что в Maple константа I представляет мнимую единицу, то, чтобы избежать путаницы при дальнейшем программировании решения, мы не стали использовать это обозначение в уравнениях, а заменили его на K(x).

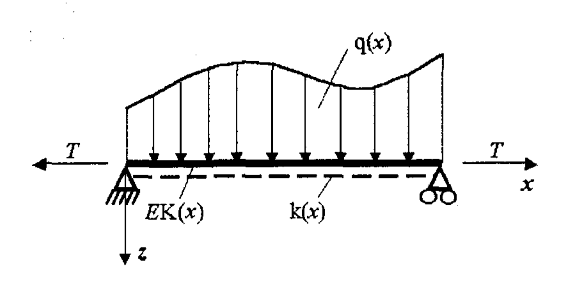


Рис. 13.1. Балка переменного сечения на сплошном упругом основании, загруженная поперечной нагрузкой и осевыми усилиями

Дифференциальное уравнение четвертого порядка можно переписать в виде системы четырех дифференциальных уравнений первого порядка

$$w'(x) = \theta(x)$$

$$\theta'(x) = \frac{M(x)}{E\,K(x)}$$

$$M'(x) = N(x)$$

$$N'(x) = q(x) + \frac{T\,M(x)}{E\,K(x)} - k(x)\,w(x),$$

которую для удобства дальнейшего использования представим в матричной форме

$$W'(x) = A(x)\,W(x) + B(x),$$

где W(x) и B(x) представляют векторы-столбцы размерности (4×1), а A(x) — квадратную матрицу размерности (4×4):

$$W(x) = \begin{bmatrix} w(x) \\ \theta(x) \\ M(x) \\ N(x) \end{bmatrix}, \quad A(x) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{E\,K(x)} & 0 \\ 0 & 0 & 0 & 1 \\ -k(x) & 0 & \frac{T}{E\,K(x)} & 0 \end{bmatrix}, \quad B(x) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ q(x) \end{bmatrix}.$$

Все неизвестные функции системы дифференциальных уравнений, представляющие компоненты вектора W(x), имеют механический смысл: w(x) — прогиб балки, θ(x) — угол поворота поперечного сечения, M(x) — изгибающий момент в сечении и N(x) — перерезывающая сила.

Для корректной постановки задачи изгиба балки следует также задать краевые условия на концах балки x=0 и x=l, где l — длина балки. Однако мы не

будем здесь выписывать их для общего случая закрепления концов, а при решении конкретных задач изгиба балок укажем условия, накладываемые на неизвестные функции в зависимости от закрепления их концов.

Теперь перейдем к описанию непосредственно самого метода начальных параметров, который мы предполагаем реализовать в Maple.[1] Вся балка по длине разбивается на n участков точками $x_0 < x_1 < \ldots < x_{n-1} < x_n$, причем $x_0=0$, $x_n=l$. Если все $\Delta_k = x_{k+1}-x_k$, $k=0..n-1$ достаточно малы по сравнению с длиной балки l, то можно пренебречь изменениями матриц A(x) и B(x) на этих интервалах и положить

$$A(x) = A(x_k) = A_k, \quad B(x) = B(x_k) = B_k, \quad \text{когда} \quad x_k \le x < x_{k+1}.$$

Тогда на интервале $x_k \le x < x_{k+1}$ поведение механической системы может быть описано системой дифференциальных уравнений с постоянными коэффициентами

$$W'(x) = A_k W(x) + B_k.$$

Решение этой системы представляем в виде ряда Тейлора в окрестности точки $x=x_k$ и сохраняя в нем некоторое конечное число членов ряда, можно выразить значение вектора W(x) в точке $x=x_{k+1}$ по следующей формуле:

$$W_{k+1} = W_k + \Delta_k W_k{}' + \frac{1}{2}\Delta_k^2 W_k{}'' + \frac{1}{6}\Delta_k^3 W_k{}''' + \frac{1}{24}\Delta_k^4 W_k{}''''.$$

Производные вектора решения W(x) в точке $x=x_k$ вычисляются последовательным дифференцированием системы с постоянными коэффициентами, описывающей поведение балки на рассматриваемом интервале изменения независимой переменной x:

$$W_k' = A_k W_k + B_k,$$
$$W_k'' = A_k (A_k W_k + B_k),$$
$$W_k''' = A_k^2 (A_k W_k + B_k),$$
$$W_k'''' = A_k^3 (A_k W_k + B_k).$$

Подставив вычисленные значения производных в точке $x = x_k$ в формулу для W_{k+1}, получим выражение вектора неизвестных функций системы W(x) в точке $x=x_{k+1}$ через его значения в точке $x=x_k$

$$W_{k+1} = A_k^* W_k + B_k^*,$$

где введены следующие обозначения

$$A_k^* = E + \Delta_k A_k + \frac{1}{2}\Delta_k^2 A_k^2 + \frac{1}{6}\Delta_k^3 A_k^3 + \frac{1}{24}\Delta_k^4 A_k^4,$$

---

[1] Описание метода начальных параметров заимствовано из книги В. А. Постнова [7].

$$B_k^* = (\Delta_k E + \frac{1}{2}\Delta_k^2 A_k + \frac{1}{6}\Delta_k^3 A_k^2 + \frac{1}{24}\Delta_k^4 A_k^3)\, B_k.$$

Через E обозначена единичная матрица размерности (4×4).

Используя полученную зависимость вектора решения $W(x)$ на правом конце k-го интервала через его значение на левом конце, можно выразить значение вектора решения в точке $x=x_{k+1}$ через его значение в начальной точке $x=x_0$, т. е. на левом конце:

$$W_k = A_k^+ W_0 + B_k^+,$$
$$A_k^+ = A_k^* A_{k-1}^+, \quad A_0^+ = A_0^*,$$
$$B_k^+ = A_k^* B_{k-1}^+ + B_k^*, \quad B_0^+ = B_0^*, \quad k = 1..n.$$

При $k=n$ это уравнение называется уравнением переноса граничных условий с начала на конец интервала изменения независимой переменной x. Оно позволяет определить значение вектора неизвестных функций $W(x)$ в любой k-й точке разбиения длины балки как функцию, зависящую от параметров внешней нагрузки и значения вектора $W(x)$ в точке $x=0$, соответствующей левому концу балки. Для определения четырех компонентов вектора W_0 следует использовать четыре граничных условия — по два на каждом конце балки, причем при вычислении граничных условий на правом конце следует использовать уравнение переноса граничных условий. После нахождения компонентов вектора W_0 из полученной системы уравнений, определение значений компонентов вектора $W(x)$ в точках разбиения балки по ее длине не представляет сложности — достаточно последовательно для $k=1..n$ применить полученное выше уравнение переноса.

Только что описанный алгоритм метода начальных параметров можно рассматривать как традиционное использование метода построения решения системы дифференциальных уравнений разложением в ряд Тейлора в условиях ручного счета или при его реализации в традиционных системах программирования, не предназначенных для работы с аналитическими вычислениями, например, Visual Fortran или C.

На первый взгляд алгоритм метода начальных параметров позволяет получить дискретное решение в точках, разбивающих балку по ее длине на заданное число интервалов. Для получения решения в промежуточных точках следует уменьшать интервалы разбиения и снова строить уравнение переноса граничных условий для большего числа точек разбиения. Однако если использовать возможности системы аналитических вычислений раскладывать в ряды функции, то метод начальных параметров позволяет получить аналитическое (в форме степенного ряда) решение, с помощью которого можно вычислить требуемые для расчета балки величины не на дискретном множестве точек, а в любой точке по длине балки.

Разложим коэффициенты системы дифференциальных уравнений вместе с вектором $W(x)$ в ряды Маклорена:

$$\mathrm{A}(x) = \sum_{i=0}^{\infty} A_i x^i ,$$

$$\mathrm{B}(x) = \sum_{i=0}^{\infty} B_i x^i ,$$

$$\mathrm{W}(x) = \sum_{i=0}^{\infty} \mathrm{W}_i x^i .$$

В них A_i являются числовыми матрицами размерности (4×4), а B_i и W_i — числовыми векторами размерности (4×1). Компоненты матриц A_i и векторов B_i известны и определяются из вида соответствующих компонентов матрицы $\mathrm{A}(x)$ и вектора $\mathrm{B}(x)$, тогда как значения компонентов векторов W_i являются неизвестными величинами, которые следует определить исходя из дифференциального уравнения задачи и удовлетворения заданным граничным условиям.

Подставив в дифференциальное уравнение разложения в степенные ряды вектора решения и ее коэффициентов, получим равенство двух степенных рядов

$$\sum_{j=0}^{\infty} (j+1)\, \mathrm{W}_{j+1}\, x^j = \sum_{j=0}^{\infty} \left(\left(\sum_{k+l=j} A_k\, \mathrm{W}_l \right) + B_j \right) x^j .$$

Приравнивая коэффициенты при одинаковых степенях переменной x, получим систему линейных уравнений относительно неизвестных коэффициентов в разложении решения $\mathrm{W}(x)$

$$\mathrm{W}_1 = A_0\, \mathrm{W}_0 + B_0 ,$$

$$2\, \mathrm{W}_2 = A_1\, \mathrm{W}_0 + A_0\, \mathrm{W}_1 + B_1 ,$$

$$3\, \mathrm{W}_3 = A_2\, \mathrm{W}_0 + A_1\, \mathrm{W}_1 + A_0\, \mathrm{W}_2 + B_2 ,$$

. . . .

из которой путем последовательных подстановок в правые части уравнений выражений для векторов W_j, начиная с первого, можно выразить все неизвестные коэффициенты в разложении решения через вектор W_0. Это позволит представить решение дифференциального уравнения изгиба балки в виде

$$\mathrm{W}(x) = \mathrm{W}_0 \left(\sum_{j=1}^{\infty} \mathrm{W}_j^{*} x^j \right) + \left(\sum_{j=1}^{\infty} B_j^{*} x^j \right),$$

где матрицы W_j^{*} и векторы B_j^{*} представляются выражениями, в которые входят известные матрицы A_k и векторы B_l.

Завершает решение задачи определение четырех компонентов вектора W_0 из системы четырех линейных уравнений, получающейся из условия удовле-

творения полученного решения граничными условиями (по два на каждом конце балки).

При описании алгоритма построения решения мы не затронули вопрос об интервале сходимости используемых рядов. Пока предположим, что все они сходятся на интервале [0, *l*] изменения независимой переменной *x*, а в дальнейшем покажем, как можно строить решение, если радиус сходимости степенных рядов меньше длины балки.

Реализовать предложенный алгоритм в традиционном языке программирования, конечно, можно, но потребуются достаточно большие временные затраты. С помощью языка Maple это делается быстро, программа состоит всего из десятка операторов. Для начала мы реализуем первую часть алгоритма, в которой неизвестные коэффициенты решения в форме степенного ряда выражаются через коэффициент W_0, представляющий свободный член ряда, т. е. по существу, мы получим уравнение переноса граничных условий с левого на правый конец. Программа, выполняющая указанные действия, представлена в примере 13.1.

Пример 13.1. Реализация уравнения переноса граничных условий

```
> # Формирование уравнения переноса граничных условий
> restart: with(linalg):
# Задание исходных данных
  alpha:=0.5:  ll:=3:  E:=1:  q:=x:  K:=(1-4*alpha/6^2*x*(6-x)):
# Количество удерживаемых членов в разложении решения
  n:=20:
# Матрица системы и вектор свободных членов
  A:=array(1..4,1..4,[[0,1,0,0],[0,0,1/(E*K),0],[0,0,0,1],[0,0,0,0]]):
  B:=array(1..4,[0,0,0,q]):
# Разложение матрицы системы и вектора свободных членов в ряды
  AT:=map(taylor, A, x=0, n+1):
  AT:=map(convert, AT, polynom):
  BT:=map(taylor, B, x=0, n+1):
  BT:=map(convert, BT, polynom):
# Формирование ряда решения с неопределенными коэффициентами
  for j from 0 to n do
    W[j]:=[W[j,1],W[j,2],W[j,3],W[j,4]];
  end do:
  WT:=sum('W[j]*(x^j)','j'=0..n):
# Вычисление правой части дифференциального уравнения
  L:=evalm(AT &* WT + BT):
# Определение коэффициентов при степенях независимой переменной
# в разложении правой части (каждый w[j] — вектор размерности 4)
  for j from 0 to n do
    w[j]:=map(coeff,L,x,j);
  end do:
```

```
# Приравнивание коэффициентов при одинаковых степенях независимой
# переменной после подстановки рядов в дифференциальное уравнение
  for c from 1 to 4 do
    for k from 1 to n do
      W[k,c]:=(w[k-1][c])/k;
    end do;
  end do:
```

В этой программе сначала мы задаем все необходимые параметры дифференциального уравнения изгиба балки: модуль упругости (E), длину балки (l1), момент инерции сечения относительно оси, проходящей через его центр тяжести (K), а также матрицу системы (A) и вектор свободных членов (B). Кроме этих величин задается количество удерживаемых в разложении решения в ряд Маклорена членов (n), так как в программе, естественно, не возможно построение ряда с бесконечным числом членов.

Далее начинается реализация непосредственно первой части алгоритма: раскладываются в ряды матрица (AT) и вектор свободных членов (BT) системы, формируется разложение вектора решения в степенной ряд (WT) с неопределенными коэффициентами, и, наконец, эти коэффициенты вычисляются по формулам, полученным путем приравнивания коэффициентов при соответствующих степенях независимой переменной в рядах левой и правой части системы после подстановки в нее всех степенных рядов. Обратим внимание, что мы не получаем явную зависимость всех неопределенных коэффициентов в ряде решения через коэффициент W_0. В Maple это делать не обязательно, памятуя о том, что если какой-либо неизвестной величине присвоить некоторое значение, то все выражения, в которые она входит, будут автоматически вычислены, чтобы соответствовать этому новому значению неизвестной величины. Поэтому, как только мы, например, присвоим какому-либо компоненту вектора W_0 значение, то будет инициирована "цепная" реакция вычисления всех коэффициентов W_1, W_2, ..., W_n, а так как W_1 будет выражен только через W_0, то W_2 также окажется вычисленным через W_0 и т. д. Если же после выполнения всех операторов примера 13.1 распечатать неопределенные коэффициенты, то мы увидим, что они выражены не только через первый компонент W_0, но и через все предшествующие. Подобного "трюка" невозможно увидеть ни в одном языке программирования!

Теперь нам остается только завершить решение задачи, удовлетворив заданным граничным условиям. Предположим, что балка жестко заделана на обоих концах. В этом случае, в соответствии с матричной формой записи системы дифференциальных уравнений изгиба балки, должны выполняться следующие условия:

$$W[1](0) = 0,\ W[2](0) = 0,$$

$$W[1](l) = 0,\ W[2](l) = 0,$$

где индекс в квадратных скобках означает соответствующий компонент вектора решения, а в круглых скобках задано значение независимой переменной, при котором выполняются эти условия.

В точке $x=0$ построенный нами вектор решения (WT) равен вектору W_0 (в программе W[0]), поэтому для выполнения первых двух условий следует положить равными нулю два первых его элемента. Как только это будет выполнено, то автоматически будут снова вычислены все коэффициенты W[j], и вектор решения WT станет зависеть только от двух неизвестных величин — элементов W[0,2] и W[0,3] вектора W[0] и, конечно, от независимой переменной x.

Для удовлетворения оставшимся двум граничным условиям на правом конце балки следует вычислить первый и второй компоненты вектора решения WT при $x=l$, приравнять их нулю и из полученной системы двух уравнений определить оставшиеся два неизвестных компонента вектора W[0]. Эта часть алгоритма представлена в примере 13.2.

Пример 13.2. Удовлетворение граничным условиям

```
> W[0,1]:=0:
  W[0,2]:=0:
  eq:=eval(WT[1],x=l1)=0;
  eq1:=eval(WT[2],x=l1)=0;
  solve({eq,eq1},{W[0,4],W[0,3]});
  assign(%):
```

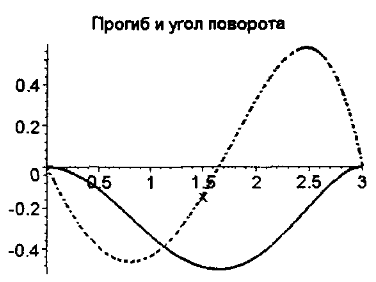
$$eq := 3.588836151 + 7.126452346\ W_{0,4} + 6.238362271\ W_{0,3} = 0$$

$$eq1 := 6.370910941 + 4.712666918\ W_{0,3} + 7.899638482\ W_{0,4} = 0$$

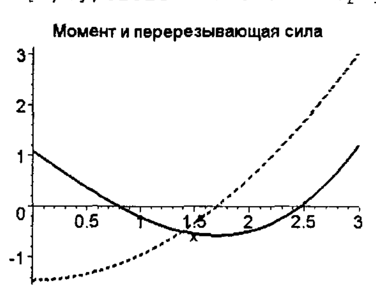
$$\{ W_{0,3} = 1.086344065, W_{0,4} = -1.454558801 \}$$

Обратите внимание на два уравнения, которые решаются командой solve(). В них действительно только две неизвестных величины — третий и четвертый компонент вектора W[0]. Как только им будут присвоены командой assign(%) значения, полученные в результате решения системы линейных уравнений, автоматически будет вычислено наше решение WT, в котором останется одна неизвестная величина — независимая переменная x. Теперь можно отобразить графики построенного решения, которые соответствуют прогибу и углу поворота сечения балки (первые два компонента вектора решения) и моменту и перерезывающей силе в сечениях по длине балки (третий и четвертый компоненты вектора решения):

```
> plot([-WT[1],-WT[2]],x=0..l1, color=black, thickness=3,
              linestyle=[1,4],title="Прогиб и угол поворота");
```

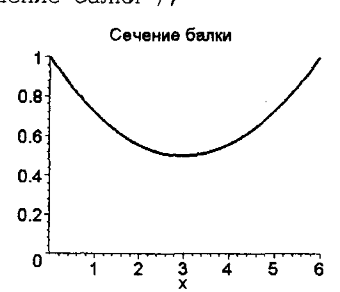



```
> plot([WT[3],WT[4]],x=0..11, color=black, thickness=3,
            linestyle=[1,4],title="Момент и перерезывающая сила");
```



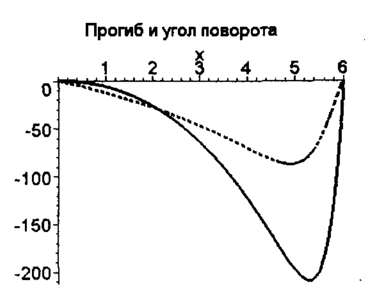
В результате разработки алгоритма модифицированного метода начальных параметров нами решена задача изгиба защемленной на обоих концах балки длиной 3 м, нагруженной распределенной гидростатической нагрузкой $q=x$ и сечение которой изменяется по следующему закону:

```
> plot(K,x=0..11, 0..1, color=black,thickness=3,linestyle=1,
            title="Сечение балки");
```



Давайте теперь увеличим длину балки до 6 м и рассчитаем ее с теми же параметрами нагружения и нарисуем графики прогиба и поворота сечения. Для этого следует снова выполнить программы примеров 13.1 и 13.2 с заменой оператора `11:=3` на оператор `11:=6` и отобразить требуемые графики командой `plot()`:

```
> plot([-WT[1],-WT[2]],x=0..11, color=black, thickness=3,
            linestyle=[1,4],title="Прогиб и угол поворота");
```



Графики прогиба и угла поворота получились не совсем такими, как ожидалось. Они должны быть очень похожи на графики соответствующих величин предыдущего примера, а у нас получилось, что даже угол поворота не равен нулю в точке максимума прогиба. В чем дело?

А вот в этом примере и проявили себя степенные ряды, в которые мы раскладывали переменные коэффициенты уравнения. Один из коэффициентов представлен функцией

$$f(x) = \frac{1}{\left(1 - \frac{1}{18}x(6-x)\right)E},$$

и его ряд Маклорена сходится на открытом интервале $(-3\sqrt{2}, 3\sqrt{2})$, а поэтому его использование для расчетов по всей длине балки и привело к неверному решению.

Выход из создавшегося затруднения прост — следует разбить балку на интервалы такой длины, в пределах которых используемые нами степенные ряды сходятся, а применить модифицированный метод начальных параметров на каждом из участков со "склейкой" решений на границах участков. Так как все неизвестные функции системы дифференциальных уравнений должны быть непрерывны по всей длине балки, то склеивание решений на границе любых двух участков должно удовлетворять следующему условию:

$$W^{(k)}(l_k) = W^{(k+1)}(0).$$

В этом равенстве верхний индекс у решения W соответствует номеру участка, на котором оно построено, l_k обозначает длину k-го участка разбиения. Модифицированная программа расчета представлена в примере 13.3.

Замечание

В связи с тем, что на каждом участке интегрирования используется местная система координат с началом в точке левой границы участка, то на каждом участке необходимо преобразовывать все переменные коэффициенты дифференциального уравнения с учетом такого сдвига начала системы координат.

Пример 13.3. Модифицированная программа расчета балки по методу начальных параметров

```
> restart:with(linalg):
# Задание количества участков и параметра в функции момента
# инерции сечения,fkrb
  n_u:=2: alpha:=0.5:
# Количество удерживаемых членов в разложении решения
  n:=20:
# Задание параметров на участках разбиения
#  ST1:=[0,0,0,0]:  ST2:=[0,0,0,0]:
  E[1]:=1: E[2]:=1:           # Модуль упругости
  q[1]:=x: q[2]:=x+3:         # Распределенная нагрузка
  l[1]:=3: l[2]:=3:           # Длины участков
  K[1]:=(1-4*alpha/6^2*x*(6-x)):    # Осевые моменты инерции сечения
  K[2]:=(1-4*alpha/6^2*(x+1.5)*(6-1.5-x)):
  ll:= l[1]+l[2]              # Длина балки
# Матрица системы и вектор свободных членов на участках разбиения
  for i from 1 to n_u do
      A||i:=
            array(1..4,1..4,
            [[0,1,0,0],[0,0,1/(E[i]*K[i]),0],[0,0,0,1],[0,0,0,0]]):
      B||i:=array(1..4,[0,0,0,q[i]]);
  end do:
# Цикл по участкам разбиения
  for i from 1 to n_u do
# Разложение матрицы системы и вектора свободных членов в ряды
      AT||i:=map( taylor,A||i,x=0,n+1);
      AT||i:=map(convert,AT||i,polynom);
      BT||i:=map(taylor,B||i,x=0,n+1);
      BT||i:=map(convert,BT||i,polynom);
      for j from 0 to n do
            W||i[j]:=[W||i[j,1],W||i[j,2],W||i[j,3],W||i[j,4]];
      end do;
      WT||i:=sum('W||i[j]*(x^j)','j'=0..n);
# Вычисление правой части дифференциального уравнения
      L:=evalm(AT||i &* WT||i+BT||i);
# Определение коэффициентов при степенях независимой переменной
# в разложении правой части (каждый w||i[j] — вектор размерности 4)
      for j from 0 to n do
            w||i[j]:=map(coeff,L,x,j);
      end do;
# Приравнивание коэффициентов при одинаковых степенях независимой
# переменной после подстановки рядов в дифференциальное уравнение
      for c from 1 to 4 do
```

```
            for k from 1 to n do
                  W||i[k,c]:=(w||i[k-1][c])/k;
            end do;
      end do;
# Склейка решений на границах участков
      if I<>n_u then
            for j from 1 to 4 do
                  W||(i+1)[0,j]:=eval(WT||i[j],x=l[i]);
#                 W||(i+1)[0,j]:=eval(WT||i[j]+ST||i[j],x=l[i]);
            end do;
      end if:
  end do:  # Завершение цикла по числу участков
# Удовлетворение граничным условиям
  W1[0,1]:=0:
  W1[0,2]:=0:
  eq:=eval(WT||n_u[1],x=l[n_u])=0;
  eq1:=eval(WT||n_u[2],x=l[n_u])=0;
  solve({eq,eq1},{W1[0,4],W1[0,3]});
  assign(%):
# Формирование решения по всей длине балки
  WT:=piecewise(x>=0 and x<l[1], WT1,
            x>=l[1] and x<=l[1]+l[2], subs(x=x-l[1], WT2)):
```

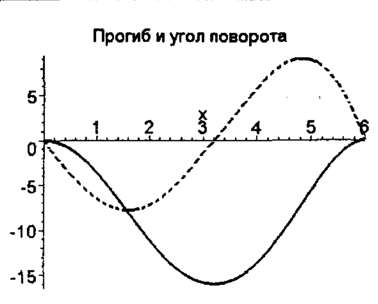
Программа примера 13.3 отличается от программы, составленной из примеров 13.1 и 13.2, тем, что исходные данные для каждого участка задаются в соответствующих массивах, а также в нее включен цикл по заданному числу участков разбиения. Расчет по каждому участку завершается склейкой решения со следующим участком в точке их соприкосновения. Завершается новый вариант программы расчета удовлетворением граничным условиям и формированием решения по всей длине балки из построенных на каждом участке, причем на участках, кроме первого, осуществляется преобразование решения к общей системе координат задания балки (координатная система первого участка).

Замечание

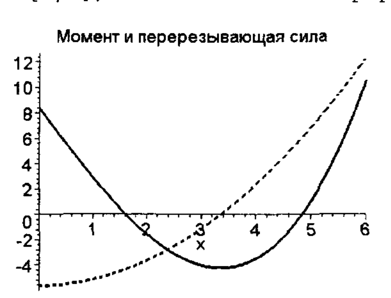
Пока не обращайте внимания на закомментированные операторы задания элементов массивов ST1 и ST2, а также на закомментированный второй оператор в теле цикла склейки решений. Они нам потребуются немного дальше.

После выполнения программы примера 13.3, в которой решается задача изгиба балки переменного сечения длиной 6 м, будут получены достоверные результаты:

```
> plot([-WT[1],-WT[2]],x=0..11, color=black, thickness=3,
            linestyle=[1,4],title="Прогиб и угол поворота");
```




```
> plot([WT[3],WT[4]],x=0..11, color=black, thickness=3,
          linestyle=[1,4],title="Момент и перерезывающая сила");
```



Если из анализа интервала сходимости рядов потребуется разбить балку на большее число участков, то следует просто добавить очередные элементы в массивы упругости, распределенной нагрузки, длин участков и осевых моментов инерции, а также задать требуемое число участков и изменить оператор вычисления длины балки.

Мы разработали алгоритм метода начальных параметров с разбиением на участки в связи с проблемой сходимости степенных рядов, однако введение интервалов разбиения позволяет расширить класс решаемых задач, введя в рассмотрение сосредоточенные нагрузки и моменты, а также распределенные нагрузки, действующие не по всей длине балки, а на отдельных ее участках.

В этом случае сначала балка разбивается на участки таким образом, чтобы сосредоточенные силы и моменты действовали в точках границ участков, и точки начала и окончания действия распределенной нагрузки также приходились на начало и конец каких-либо участков, причем распределенная нагрузка может действовать на нескольких идущих подряд участках. После этого следует исследовать сходимость степенных рядов, в которые раскладываются переменные коэффициенты системы дифференциальных уравнений на каждом полученном участке. Если найдутся участки, длины которых больше интервала сходимости рядов, то такие участки следует разбить на более мелкие, в которых ряды будут сходиться. На этом разбиение балки на участки можно считать завершенным.

С введением в рассмотрение сосредоточенных сил и моментов следует несколько изменить условия склейки решений на границах участков. Если на границе двух участков действует сосредоточенная сила Q, то при склейке решений ее следует учитывать в четвертом элементе вектора решений

$$W^{(k)}[4](l_k) = W^{(k+1)}[4](0) + Q,$$

а если сосредоточенный момент M, то в третьем

$$W^{(k)}[3](l_k) = W^{(k+1)}[3](0) + M.$$

Остальные два элемента вектора решений склеиваются по старой схеме их непрерывного изменения при переходе через границу соседних участков

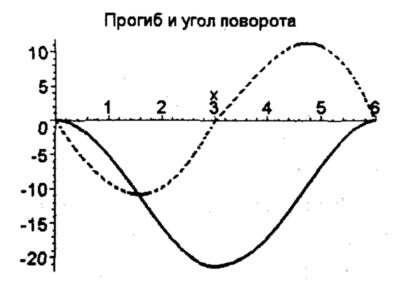
$$W^{(k)}[1](l_k) = W^{(k+1)}[1](0),$$
$$W^{(k)}[2](l_k) = W^{(k+1)}[2](0).$$

Для реализации этого алгоритма в программе примера 13.3 следует снять комментарии с упомянутых в замечании операторов. Массивы `ST1`, `ST2` и т. д. содержат в третьих элементах значения сосредоточенных моментов, а в четвертых — сосредоточенных сил, действующих на границах участков. Их первые два элемента всегда равны нулю. Если на границе участков сосредоточенной силы или момента нет, то соответствующие элементы равны нулю. Целое число в имени массива показывает, что в нем хранятся значения сосредоточенных моментов и сил между участком с номером, соответствующим целому числу и следующим за ним участком. Например, если в нашей задаче действует сосредоточенный момент и сила по центру балки, то следует задать один массив

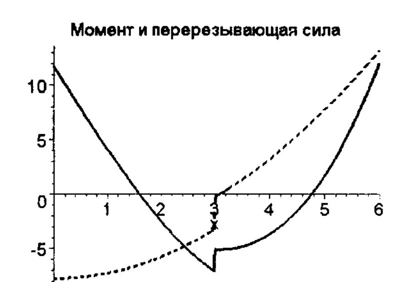
```
> ST1:=[0, 0, 2, 3];
```

В этом случае эпюры прогиба, угла поворота сечения, момента и перерезывающей силы будут выглядеть так:

```
> plot([-WT[1],-WT[2]],x=0..11, color=black, thickness=3,
         linestyle=[1,4],title="Прогиб и угол поворота");
```




```
> plot([WT[3],WT[4]],x=0..11, color=black, thickness=3,
         linestyle=[1,4],title="Момент и перерезывающая сила");
```



Они полностью согласуются с действующей нагрузкой: в точке действия сосредоточенного момента эпюра угла поворота сечения имеет слом, а в эпюре момента наблюдается скачок, равный величине сосредоточенного момента, в эпюре перерезывающей силы скачок на величину сосредоточенной силы наблюдается в точке ее приложения.



ГЛАВА 14

Задачи теории упругости

Первой задачей, которую мы рассмотрим, будет задача определения напряженного состояния в точке деформируемого тела — совокупности напряжений, действующих по всевозможным площадкам, проведенным через эту точку.

В общем случае тело, находящееся под воздействием внешних сил, находится в объемном напряженном состоянии. Такое состояние количественно описывается симметричным тензором напряжений третьего ранга с шестью независимыми компонентами. Если на всех площадках, параллельных одной и той же плоскости, напряжения отсутствуют или настолько малы, что ими можно пренебречь, то напряженное состояние такого тела будет характеризоваться симметричным тензором второго ранга с тремя независимыми компонентами. Подобное напряженное состояние называется плоским. В таком состоянии, например, будет находиться тонкая пластинка под воздействием произвольной системы сил, приложенных к кромкам пластинки и лежащих в ее плоскости.

Задача 14.1

Исследовать напряженное состояние в точке пластинки, находящейся в условиях плоского напряженного состояния.

Решение. Вырежем элементарный параллелепипед из пластинки в окрестности произвольной точки сечениями, перпендикулярными плоскости пластинки. Со стороны среды, окружающей параллелепипед, на него в общем случае действуют как нормальные, так и касательные усилия. На рис. 14.1, *а* показаны векторы нормальных и касательных напряжений, соответствующие этим усилиям. Внутри параллелепипеда напряженное состояние считается однородным.

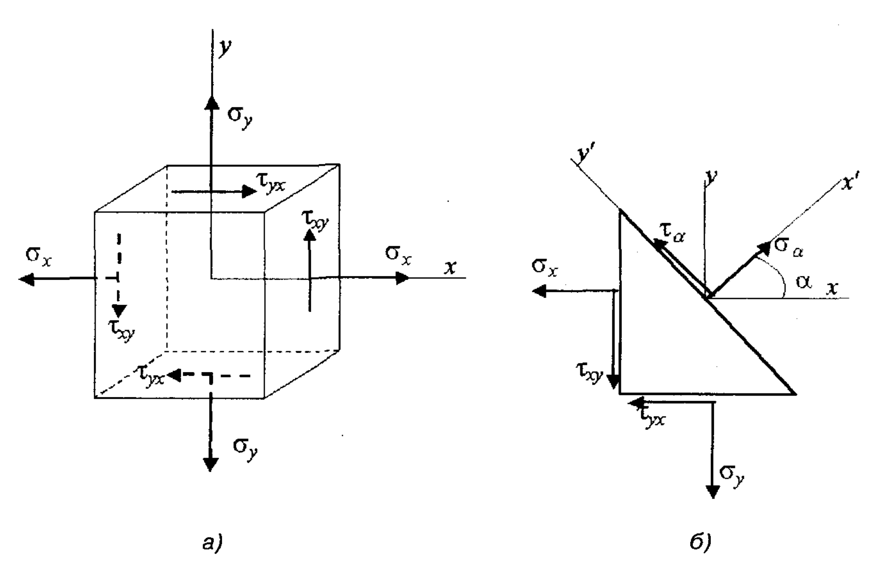


Рис. 14.1. Напряжения на гранях элемента, находящегося в условиях плоского напряженного состояния

Нормальное σ_α и касательное τ_α напряжения на наклонной площадке, проходящей через внутреннюю точку элементарного параллелепипеда под углом α к оси x и перпендикулярной ее ненагруженным граням (рис. 14.1, *б*), определяется по известным формулам:

$$\sigma_\alpha = \frac{1}{2}\sigma_x + \frac{1}{2}\sigma_y + \frac{1}{2}(\sigma_x - \sigma_y)\cos(2\alpha) + \tau_{xy}\sin(2\alpha)$$

$$\tau_\alpha = -\frac{1}{2}(\sigma_x - \sigma_y)\sin(2\alpha) + \tau_{xy}\cos(2\alpha)$$

Конечно, вычисление напряжений по этим формулам не представляет труда, и это можно выполнить на любом калькуляторе, не прибегая к такой мощной системе, как система аналитических вычислений Maple.

Мы используем эти формулы для создания анимационной картинки, в которой вычисляются напряжения на гранях поворачивающегося элементарного параллелепипеда, расположенного внутри исходного элементарного параллелепипеда с заданными на его гранях напряжениями. Это позволит нам наблюдать изменение нормального и касательного напряжений на наклонной площадке.

Прежде всего разработаем процедуру `nds()`, создающую графический образ внешнего и внутреннего параллелепипеда, повернутого на угол α, а также рассчитывающую и отображающую напряжения на гранях обоих параллелепипедов. Ее текст представлен в примере 14.1.

Пример 14.1. Вычисление напряжений на наклонной площадке

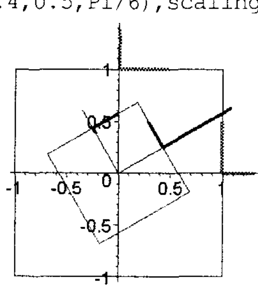
```
> nds:=proc(Sx,Sy,Txy,alpha)
      local out,inn,x1,y1,sx,sx1,sy,sy1,t,t1,t2;
# Исходный параллелепипед
      out:=`plottools/rectangle`([-1,1],[1,-1],scaling=constrained);
# Отображение напряжения на правой вертикальной и верхней горизонтальной
# гранях исходного параллелепипеда
      sxout:=plot([[1,0],[1+Sx,0]],color=red,thickness=4);
      toutvert:=plot([[1,0],[1,0+Txy]],color=red,thickness=4);
            syout:=plot([[0,1],[0,1+Sy]],color=red,thickness=4);
      tinhor:=plot([[0,1],[0+Txy,1]],color=red,thickness=4);
      out:=`plots/display`({out,sxout,syout,toutvert,tinhor});
# Внутренний параллелепипед
      inn:=`plottools/rectangle`([-0.5,0.5],[0.5,-0.5],
            scaling=constrained);
# Локальные оси координат внутреннего параллелепипеда
      x1:=plot([[0,0],[0.7,0]],color=black,thickness=1);
      y1:=plot([[0,0],[0,0.7]],color=black,thickness=1);
# Нормальное напряжение на правой вертикальной грани
# внутреннего параллелепипеда
      sx:=(Sx+Sy)/2+(Sx-Sy)/2*cos(2*alpha)+Txy*sin(2*alpha);
      sx1:=plot([[0.5,0],[0.5+sx,0]],color=red,thickness=3);
# Касательное напряжение на правой вертикальной грани
# внутреннего параллелепипеда
      t:=-(Sx-Sy)/2*sin(2*alpha)+Txy*cos(2*alpha);
      t1:=plot([[0.5,0],[0.5,0+t]],color=magenta,thickness=3);
# Нормальное напряжение на верхней горизонтальной грани
# внутреннего параллелепипеда
   sy:=(Sx+Sy)/2+(Sx-Sy)/2*cos(2*(alpha+Pi/2))+Txy*sin(2*(alpha+Pi/2));
      sy1:=plot([[0,0.5],[0,0.5+sy]],color=red,thickness=3);
# Касательное напряжение на верхней горизонтальной грани
# внутреннего параллелепипеда
      t2:=plot([[0,0.5],[0+t,0.5]],color=magenta,thickness=3);
# Вычерчивание внутреннего параллелепипеда и напряжений на его гранях
      inn:=`plots/display`({inn,x1,y1,sx1,sy1,t1,t2});
# Поворот внутреннего параллелепипеда на угол alpha
      inn:=`plottools/rotate`(inn,alpha);
# Отображение внешнего и внутреннего
# повернутого на угол alpha параллелепипедов
      `plots/display`({out,inn},scaling=constrained);
      end proc:
```

При обращении к процедуре `nds()` следует задать компоненты тензора напряжения на гранях исходного элементарного параллелепипеда в декартовой

системе координат и угол поворота внутреннего параллелепипеда, напряжения на гранях которого требуется определить.

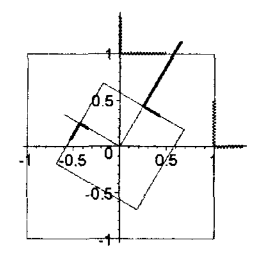
Для отображения полученного с помощью процедуры `nds()` графического образа следует использовать команду `display()`. Например, следующая команда отобразит напряжения на гранях внутреннего параллелепипеда, повернутого на угол $\alpha=30°$, если на гранях внешнего элементарного параллелепипеда действуют нормальные $\sigma_x = 0{,}3$ МПа, $\sigma_y = 0{,}4$ МПа и касательное $\tau_{xy} = 0{,}5$ МПа напряжения:

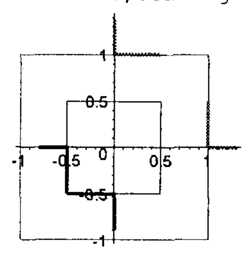
```
> plots[display](nds(0.3,0.4,0.5,Pi/6),scaling=constrained);
```

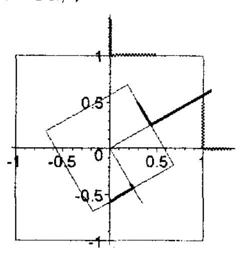


Для создания анимации, в которой внутренний параллелепипед поворачивается на 360° и при этом отображаются изменяющиеся с поворотом векторы напряжений на его гранях, следует создать последовательность графических образов повернутого на разные углы внутреннего параллелепипеда и отобразить ее командой `display()` с опцией `insequence=true`:

```
> plots[display]([seq(nds(0.3,0.4,-0.7,2*Pi/90*i),i=0..90)],
                        insequence=true,scaling=constrained);
```







Замечание

На рисунке представлены кадры анимации, соответствующие повороту внутреннего параллелепипеда на 60°, 180° и 300°.

Известно, что в элементарном параллелепипеде существуют площадки, на которых нормальные напряжения принимают экстремальные значения. Причем если на какой-либо площадке нормальное напряжение максимально, то на перпендикулярной к ней оно минимально и наоборот. Такие площадки называются *главными*. Они характеризуются еще тем, что на них ка-

сательные напряжения отсутствуют. Если наблюдать за напряжениями одной грани вращающегося внутреннего параллелепипеда, то действительно можно заметить, что при определенных углах поворота касательное напряжение на ней будет нулевым, а нормальное напряжение будет принимать либо максимальное, либо минимальное значение. Для точного определения положения главных площадок следует найти точки, в которых нормальное напряжение как функция переменной угла поворота достигает экстремума. Такими точками могут быть только точки, в которых производная этой функции равна нулю:

```
> sx:=(alpha)->(sigma[x]+sigma[y])/2+(sigma[x]-sigma[y])/2*cos(2*alpha)+
              tau[x*y]*sin(2*alpha);
```

$$sx := \alpha \to \frac{1}{2}\sigma_x + \frac{1}{2}\sigma_y + \frac{1}{2}(\sigma_x - \sigma_y)\cos(2\,\alpha) + \tau_{xy}\sin(2\,\alpha)$$

```
> _EnvAllSolutions:=true:
> main:=solve(diff(sx(alpha),alpha)=0,alpha);
```

$$main := \frac{1}{2}\arctan\left(2\,\frac{\tau_{xy}}{\sigma_x - \sigma_y}\right) + \frac{1}{2}\pi\,_Z1\sim$$

Среди множества корней производной нормального напряжения как функции угла поворота площадки только четыре соответствуют различным площадкам. Чтобы их определить, достаточно взять четыре идущих подряд значения переменной *_Z1*, принимающей целые значения. При этом на всех получаемых поворотом на выбранные углы площадках нормальные напряжения будут либо максимальными, либо минимальными, а сами площадки будут получаться последовательным поворотом на прямой угол одной из них. Отобразим главные площадки для рассчитываемой задачи:

```
> alphaMain:=eval(main,{tau[x*y]=0.5,sigma[x]=0.3,sigma[y]=0.4,_Z1=1});
```

$$alphaMain := -.7355638370 + \frac{1}{2}\pi$$

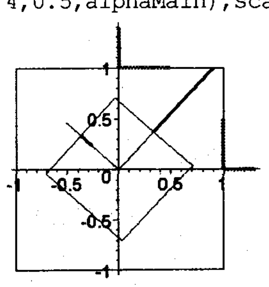
```
> tx:=-(sigma[x]-sigma[y])/2*sin(2*alpha)+tau[x*y]*cos(2*alpha);
```

$$tx := -\frac{1}{2}(\sigma_x - \sigma_y)\sin(2\,\alpha) + \tau_{xy}\cos(2\,\alpha)$$

```
> eval(tx,{alpha=alphaMain,tau[x*y]=0.5,sigma[x]=0.3,sigma[y]=0.4});
```

$$-.15\;10^{-9}$$

```
> plots[display](nds(0.3,0.4,0.5,alphaMain),scaling=constrained);
```



Здесь же мы вычислили на главной площадке касательное напряжение, которое действительно оказалось равным нулю. Из рисунка видно, что на главной площадке с выбранным углом наклона alphaMain нормальное напряжение максимально, на соседней с ней (угол поворота alphaMain+90°) минимально. На противоположных указанным площадках нормальные напряжения также будут, соответственно, максимальным и минимальным.

Аналогично нахождению экстремальных значений нормальных напряжений можно определить площадки, на которых касательное напряжение достигает своего экстремума:

```
> t:=(alpha)->-(sigma[x]-sigma[y])/2*sin(2*alpha)+tau[x*y]*cos(2*alpha);
```

$$t := \alpha \to -\frac{1}{2}(\sigma_x - \sigma_y)\sin(2\,\alpha) + \tau_{x\,y}\cos(2\,\alpha)$$

```
> T:=solve(diff(t(alpha),alpha)=0,alpha);
```

$$T := -\frac{1}{2}\arctan\left(\frac{1}{2}\,\frac{\sigma_x - \sigma_y}{\tau_{x\,y}}\right) + \frac{1}{2}\pi\,_Z3\sim$$

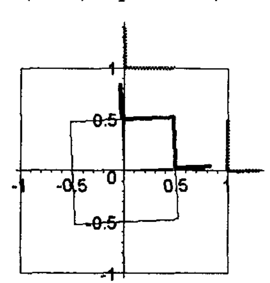
```
> alphaTau:=eval(T,{tau[x*y]=0.5,sigma[x]=0.3,sigma[y]=0.4,_Z3=0});
```

$$alphaTau := .04983432624$$

```
> eval(t(alphaTau),{tau[x*y]=0.5,sigma[x]=0.3,sigma[y]=0.4});
```

$$.5024937810$$

```
> plots[display](nds(0.3,0.4,0.5,alphaTau),scaling=constrained);
```



Можно заметить, что на площадках с максимальным касательным напряжением нормальное напряжение не равно нулю.

Задача 14.2

Определить прогибы свободно опертой прямоугольной пластинки размерами $a \times b$, вызванные распределенной нагрузкой $q=q(x,y)$.

Решение. Воспользуемся известным решением для функции прогиба $w(x,y)$ в двойных тригонометрических рядах [8]

$$w(x, y) = \left(\sum_{m=1}^{\infty}\left(\sum_{n=1}^{\infty} a_{m,n}\sin\left(\frac{m\,\pi\,x}{a}\right)\sin\left(\frac{n\,\pi\,y}{b}\right)\right)\right),$$

в котором неизвестные коэффициенты $a_{m,\,n}$ вычисляются по формуле

$$a_{m,n} = 4\,\frac{\displaystyle\int_0^a\!\!\int_0^b \mathrm{q}(x,y)\,\sin\left(\frac{m\,\pi\,x}{a}\right)\sin\left(\frac{n\,\pi\,y}{b}\right)dx\,dy}{\mathrm{D}\,\pi^4\left(\dfrac{m^2}{a^2}+\dfrac{n^2}{b^2}\right)^2 a\,b}.$$

Здесь D цилиндрическая жесткость пластины на изгиб, вычисляемая в соответствии с формулой

$$\mathrm{D} = \frac{1}{12}\,\frac{E\,h^3}{1-\nu^2},$$

в которой E и ν являются модулем упругости и коэффициентом Пуассона материала пластины, а h ее толщина.

Зная прогиб пластины, можно вычислить изгибающие M_x, M_y и крутящий H моменты по формулам

$$M_x = -\mathrm{D}\left(\left(\frac{\partial^2}{\partial x^2}\mathrm{w}(x,y)\right)+\nu\left(\frac{\partial^2}{\partial y^2}\mathrm{w}(x,y)\right)\right)$$

$$M_y = -\mathrm{D}\left(\left(\frac{\partial^2}{\partial y^2}\mathrm{w}(x,y)\right)+\nu\left(\frac{\partial^2}{\partial x^2}\mathrm{w}(x,y)\right)\right)$$

$$H = -\mathrm{D}\,(1-\nu)\left(\frac{\partial^2}{\partial y\,\partial x}\mathrm{w}(x,y)\right).$$

Разработаем процедуру вычисления прогиба пластины в зависимости от ее геометрических размеров, физико-механических характеристик и действующей нагрузки (пример 14.2).

Пример 14.2. Прогиб свободно опертой пластинки

```
> thinPlate:=proc(a,b,q,x,y,M,N,h,E,nu)
	local w,m,n,D;
	D:=1/Pi^4/a/b/E/h^3*12*(1-nu^2);
	w:=0;
	for m to M do
		for n to N do
			w:=w+(
		int(int(q*sin(m*Pi*x/a)*sin(n*Pi*y/b),x=0..a),y=0..b))*
		4/a/b/(m^2/a^2+n^2/b^2)/Pi^4*sin(m*Pi*x/a)*sin(n*Pi*y/b);
		end do;
	end do;
	w:=w*D;
	end proc:
```

В процедуру `thinPlate()` передаются геометрические размеры пластины (аргументы `a`, `b` и `h`), модуль упругости `E` и коэффициент Пуассона `nu`, выражение для действующей нагрузки `q` и имена переменных (аргументы `x` и `y`), от которых она зависит. Аргументами `M` и `N` определяется количество удерживаемых в ряде Фурье решения членов.

Теперь с помощью разработанной процедуры можно исследовать полученное в виде ряда Фурье решение. Рассчитаем прогибы и моменты квадратной свободно опертой пластины, нагруженной гидростатическим давлением q=10*x, удерживая в рядах разное количество членов:

```
> i:=0:N:=20:E:=2*10^11:nu:=0.2:q:=10*x:h:=0.1:a:=1:b:=1:
  for n from 1 to N do
   i:=i+1:
   s||i:=thinPlate(a,b,q,x,y,n,n,h,E,nu);
   ss||i:=plot3d(s||i,x=0..a,y=0..b,title=cat("n = ",convert(n,string))):
   Mx||i:=plot3d(-E*h^3/12/(1-nu^2)*(diff(s||i,x$2)+nu*diff(s||i,y$2)),
              x=0..a,y=0..b,title=cat("n = ",convert(n,string))):
   My||i:=plot3d(-E*h^3/12/(1-nu^2)*(diff(s||i,y$2)+nu*diff(s||i,x$2)),
              x=0..a,y=0..b,title=cat("n = ",convert(n,string))):
   H||i:=plot3d(-E*h^3/12/(1-nu^2)*(1-nu)*diff(s||i,x,y),
               x=0..a,y=0..b,title=cat("n = ",convert(n,string))):
 end do:
```

Замечание

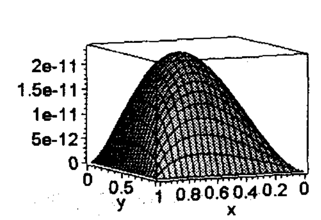
В приведенной программе двойные ряды Фурье суммируются по прямоугольникам. Это означает, что в ряде удерживаются все члены, индексы которых удовлетворяют неравенствам $m<M$, $n<N$. В нашей программе M=N. Двойные ряды можно суммировать по треугольникам, когда выполняется условие $m+n<K$, где K — некоторое заданное положительное число, а также по окружностям, когда индексы членов ряда удовлетворяют неравенству $m^2+n^2<K$.

Теперь можно посмотреть, как ведут себя решения для прогиба и моментов с разным количеством удержанных членов ряда.

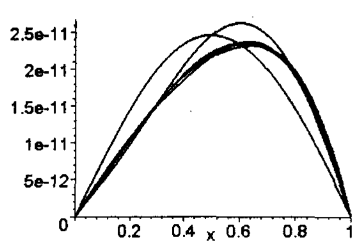
Отобразим поверхность прогиба при N=20 и прогибы в сечении $y=b/2$ при разном количестве удержанных членов (N=1..20):

```
> plots[display]([ss||20],axes=BOXED);
```

n = 20



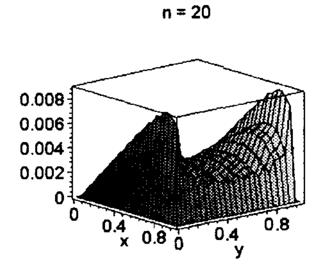
```
> plot([seq(eval(s||i,y=b/2),i=1..20)],x=0..b,color=black,thickness=2);
```



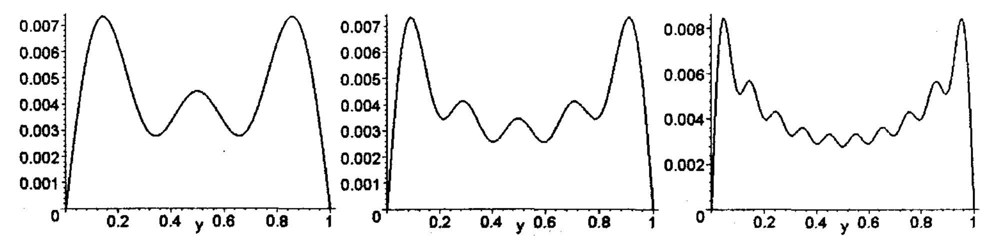
Из графика прогибов в сечении $y=b/2$ видно, что только первые два прогиба, соответствующие рядам при N=1 и N=2, достаточно сильно отличаются друг от друга и остальных прогибов. Это говорит о быстрой сходимости полученного в форме двойного ряда Фурье решения для прогиба пластины.

Теперь исследуем сходимость рядов для моментов в пластине. Они, конечно, должны сходиться медленнее ряда для прогиба, так как получены из последнего двойным дифференцированием. Построим поверхность изгибающего момента M_y и анимацию его графиков в сечении $x=0.8\ b$ при увеличении числа членов в решении:

```
> plots[display]([My20],axes=BOXED,orientation=[-40,70]);
```




```
> plots[display]([seq(plot(eval(-E*h^3/12/(1-nu^2)*(diff(s||i,y$2)+
            nu*diff(s||i,x$2)),x=0.8*a),y=0..b,color=black),i=1..20)],
            insequence=true,thickness=2);
```

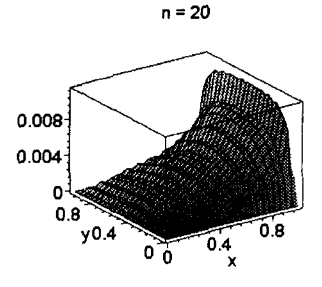


Замечание

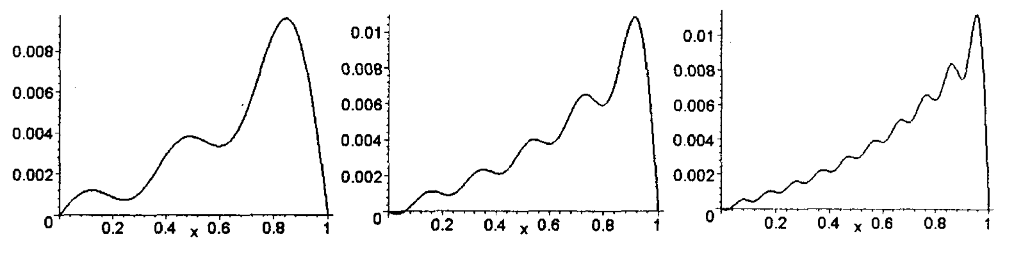
Для анимации показаны кадры, соответствующие решениям, в которых N=5, 10, 20.

Видно, что для изгибающего момента M_y даже при N=20 решение осциллирует. Аналогичная картина наблюдается и для второго изгибающего момента M_x:

```
> plots[display]([Mx20],axes=BOXED,orientation=[-125,60]);
```




```
> plots[display]([seq(plot(eval(-E*h^3/12/(1-nu^2)*(diff(s||i,x$2)+
          nu*diff(s||i,y$2)),y=b/2),x=0..a,color=black),i=1..20)],
          insequence=true,thickness=2);
```

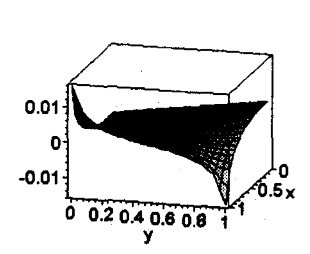


Замечание

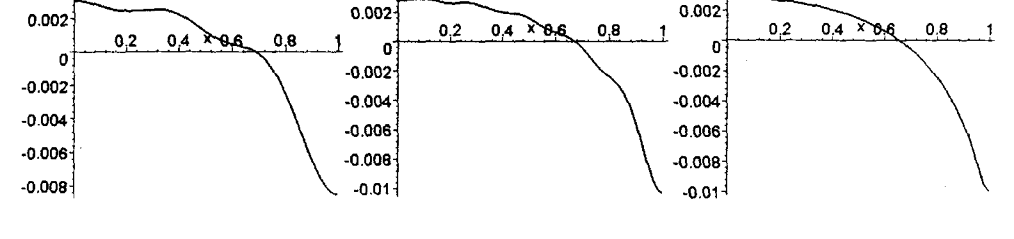
Для анимации показаны кадры, соответствующие решениям, в которых N=5, 10, 20.

Что касается крутящего момента H, то здесь картина несколько лучше, чем для изгибающих моментов. При N=20 в его графике уже отсутствуют осцилляции:

```
> plots[display]([H20],axes=BOXED,orientation=[15,70]);
```



```
> plots[display]([seq(plot(eval(-E*h^3/12/(1-nu^2)*(1-nu)*diff(s||i,x,y),
   y=0.95*b),x=0..a,color=black),i=1..20)],insequence=true,thickness=2);
```



Представленные в этой главе решения задач теории упругости могут служить основой для разработки методических материалов при изучении курса теории упругости в технических университетах.

Список литературы

1. Heal K. M., Hansen M. L., Rickard K. M. "Maple 6 Learning Guide". Waterloo Maple Inc., 2000. — 314 p.
2. Monagan M. B., Geddes K. O., Heal K. M., Labahn G., Vorkoetter S. M., McCarron J. "Maple 6 Programming Guide". Waterloo Maple Inc., 2000. — 586 p.
3. Волков Е. А. "Численные методы: Учеб. пособие для вузов". — М.: Наука, 1987. — 248 с.
4. Евсеев Е. А., Зенкевич Н. А., Лукьянова А. Е. "Основы математического анализа для менеджеров: Учебное пособие". — СПб.: Изд-во С.-Петербургского университета, 1996. — 108 с.
5. Коллатц Л., Альбрехт Ю. "Задачи по прикладной математике". — М.: Мир, 1978. — 167 с.
6. Пискунов Н. С. "Дифференциальное и интегральное исчисления". — М.: Наука, 1978. Т. II. — 576 с.
7. Постнов В. А. "Численные методы расчета судовых конструкций". — Л.: Судостроение, 1977. — 280 с.
8. Рекач В. Г. "Руководство к решению задач прикладной теории упругости". — М.: Высшая школа, 1973. — 384 с.
9. Сборник заданий для курсовых работ по теоретической механике. Под ред. проф. А. А. Яблонского. — М.: Высшая школа, 1978. — 388 с.
10. Фихтенгольц Г. М. "Курс дифференциального и интегрального исчисления". — М.: Наука, 1969. Т. II. — 800 с.

Предметный указатель

А

Анимация:
animate() 268
animate3d() 270
display() 269, 271
формат GIF 269, 271

В

Ввод/вывод:
readdata() 325
writedata() 324
буферизованный
и небуферизованный 327
дескриптор файла 327
закрытие файла, fclose(), close() 328
открытие файла, fopen() 327
открытие файла, open() 328
текущая позиция файла, filepos() 328
удаление файла, fremove() 329
файл, режим доступа 327
файл, текстовый и двоичный 327
Выделение решений, assign() 140
Вызов внешних процедур 339
define_external() 340
дескриптор данных 340
Вызов команд 86
Выражение 82
Вычисление в точке:
eval() 132
evalf() 133
evalhf() 133
subs() 132
Вычисление выражений, value() 86
Вычисление имен 128
assigned() 131
eval() 128
evaln() 130
полное 128
уровень вычисления 128

Г

Графические структуры:
AMBIENLIGHT() 262
AXESLABELS() 235
AXESSTILE() 236
AXESTICKS() 235
COLOR() 236, 263
CURVES() 235
FONT() 236
GRID() 262
GRIDSTYLE() 263
LIGHT() 263
LIGHTMODEL() 263
LINESTYLE() 236
MESH() 262
POINTS() 235
POLYGONS() 235
SCALING() 236
STYLE() 263
SYMBOL() 236
TEXT() 235
THICKNESS() 236
TITLE() 236
VIEW() 236

Графический интерфейс пользователя:
контекстная панель инструментов 23
контекстное меню 24, 42
основная панель инструментов 23
основное меню 22
палитры 32
рабочая область 24
стандартное меню рабочего листа 47
строка состояния 24
типы основного меню 45

Д

Двумерная графика:
contourplot() 230
coordplot() 226
densityplot() 229
display() 241
fieldplot() 231
gradplot() 231
imlicitplot() 227
inequal() 228
loglogplot() 228
logplot() 228
odeplot() 232
plot() 214, 235
PLOT-структура 233
plygonplot() 229
polarplot() 224
semilogplot() 228
textplot() 232
опции 214
пакет plots 224
пакет plottools 237
Дифференцирование и интегрирование:
D() 150
diff() 145
evalf() 147
int() 146

К

Команды:
add() 114
algsubs() 126
coeff() 118
convert() 127
denom() 119
evalb() 285
has() 123
hastype() 124
is() 286
isolate() 151
lhs() 119
lhs() 284
limit() 397
map() 113, 122, 296
map2() 113
member() 110
mul() 114
nops() 120
numer() 119
op() 116, 120
piecewise() 242
protect() 279
remove() 115, 122, 296
rhs() 119, 284
select() 115, 122, 296
selectremove() 115, 296
seq() 107
series() 442
simplify() 126
sort() 117
specfunc() 124
subs() 125
subsop() 126
time() 310
type() 282
unapply() 140
unprotect() 278
whattype() 120
zip() 116, 296
разделитель 71
Константы 78

М

Модули:
module() 331
инкапсуляция 331
операция экспортирования,
пакеты 331
параметр thismodule 335
реализация объектов 331

О

Ограничения на переменные:
about() 105
additionally() 103
assume() 101
coulditbe() 104
is() 104
Операнды выражения 120
Операторы:
break 295
error 303, 317
finally 319
next 295
return 316
try-catch 318
ветвления, if 290
присваивания 290
цикла, for-from 292
Операции:
$ (знак доллара) 108
if 292
intersect 110
minus 110
union 110
без операндов, %, %%, %%% 277
бинарные 276
диапазон 109
композиция двух функций, @ 277
логические 286
нейтральные 278
отношения 284
повторная композиция, @@ 277
присваивания 83
проверка типа, :: 302
унарные 275

П

Пакеты:
dotprod() 370
geom3d 380, 382
geometry 373
linalg 160
LinearAlgebra 160
powseries 445
Переменная 81
неизвестная 82
Полином 92
Преобразование выражений:
collect() 97
combine() 95
expand() 91
factor() 92
normal() 94
rationalize() 99
simplify() 87
Проверка решений:
eval() 138
map() 139
subs() 139
Пространственная графика:
contorplot3d() 260
coordplot3d() 257
cylinderplot() 255
display() 266
fieldplot3d() 261
gradplot3d() 261
implicitplot3d() 259
plot3d() 246, 262
PLOT3D-структуры 262
polygonplot3d() 261
spacecurve() 258
sphereplot() 257
textplot3d() 260
tubeplot() 259
опции 246
пакет plots 255
пакет plottools 264
Процедуры 299
возврат невычисленного значения 320
возвращаемое значение 299, 313
вычисление локальных переменных 305
локальные и глобальные переменные 304
массив args 302
неименованные 300
операнды типа 321
опции 308
передача параметров 301
сохранение в файле 323
таблица значений 309

Р

Рабочий лист 24
группа вычислений 25
Продолжение рубрики см. на с. 526

Рабочий лист *(окончание)*
область ввода 25
область ввода графики 25
область вывода 24, 29
секция 59
стандартное меню 47
форматы области вывода 29
форматы сохранения 41, 48
Решение дифференциальных уравнений:
D(), оператор 150
dsolve() 149
Решение уравнений:
fsolve() 141
isolve() 143
msolve() 143
rsolve() 143
solve() 135

С

Сложные типы данных:
массив 110
множество 108
неравенство 135
последовательность 106
список 108
таблица 112
уравнение 118, 134
Строки 79

Т

Типы данных 281
series 443

У

Устройство отображения графики 233
plotsetup() 234

Ч

Числа:
комплексные числа 77
обыкновенные дроби 74
радикалы 75
с плавающей точкой 76
целые 72

Ш

Шаблоны-заполнители 33

Э

Электронная таблица 35
влияющая ячейка 39
зависимая ячейка 39

Я

Язык Maple:
алгебраический контекст 284
алфавит 274
булевый контекст 285
выражение 281
дерево выражения 282
индексные имена 279
ключевые слова 275
лексемы 275
натуральные числа 280
символьные имена 278
строка 280
структурный тип 287
целые числа 280